PPG INSTITUTE OF TECHNOLOGY

(Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai)
Recognized by UGC | Accredited by NAAC with A | ISO Certified Institution
NH 209, Sathy Main Road, Saravanampatti, Coimbatore – 641035, Tamil Nadu

CCS342 – DEVOPS LABORATORY RECORD

DEPARTMENT OF INFORMATION TECHNOLOGY

PPG INSTITUTE OF TECHNOLOGY

(Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai)
Recognized by UGC I Accredited by NAAC with A I ISO Certified Institution
NH 209, Sathy Main Road, Saravanampatti, Coimbatore – 641035, Tamil Nadu

DEPARTMENT OF INFORMATION TECHNOLOGY

CCS342 - DEVOPS LABORATORY RECORD

NAME:.....ROLL NO:.....

SEMESTER: ..... BRANCH:.....

Certified bonafide record of work done by**.....**

PLACE: COIMBATORE. DATE:.....

Staff In-Charge **Head of the Department**

University Register Number**:.....**

Submitted for the University Practical Examination held on**.....**

INTERNAL EXAMINER **EXTERNAL EXAMINER**

TABLE OF CONTENTS

TOTAL MARKS :

FACULTY INCHARGE :

Ex. No: 1 **CREATE MAVEN BUILD PIPELINE IN AZURE**

Date:

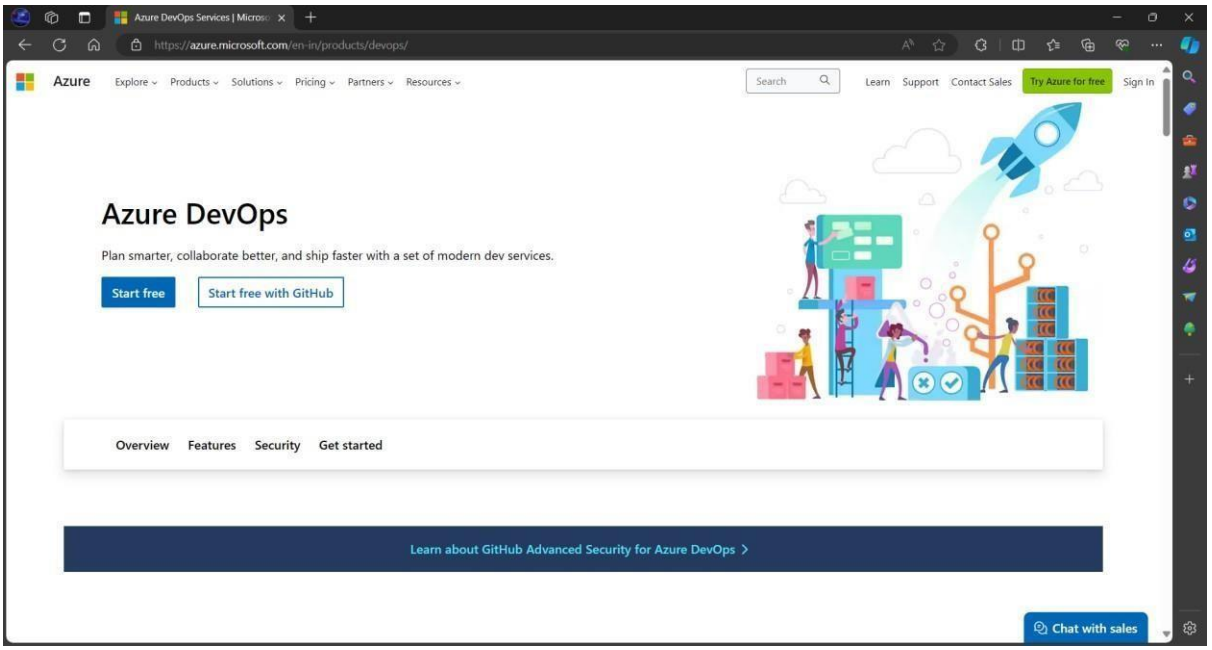
AIM

To build a maven build pipeline in Azure.

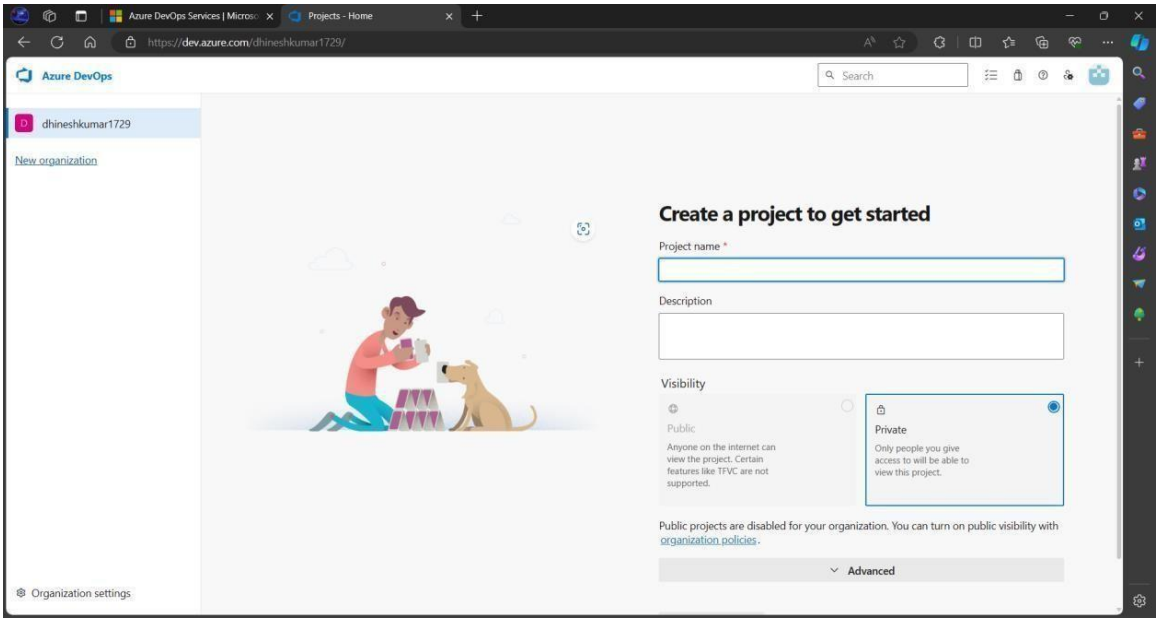
PROCEDURE

STEP 1: Create an account on the Azure DevOps website.

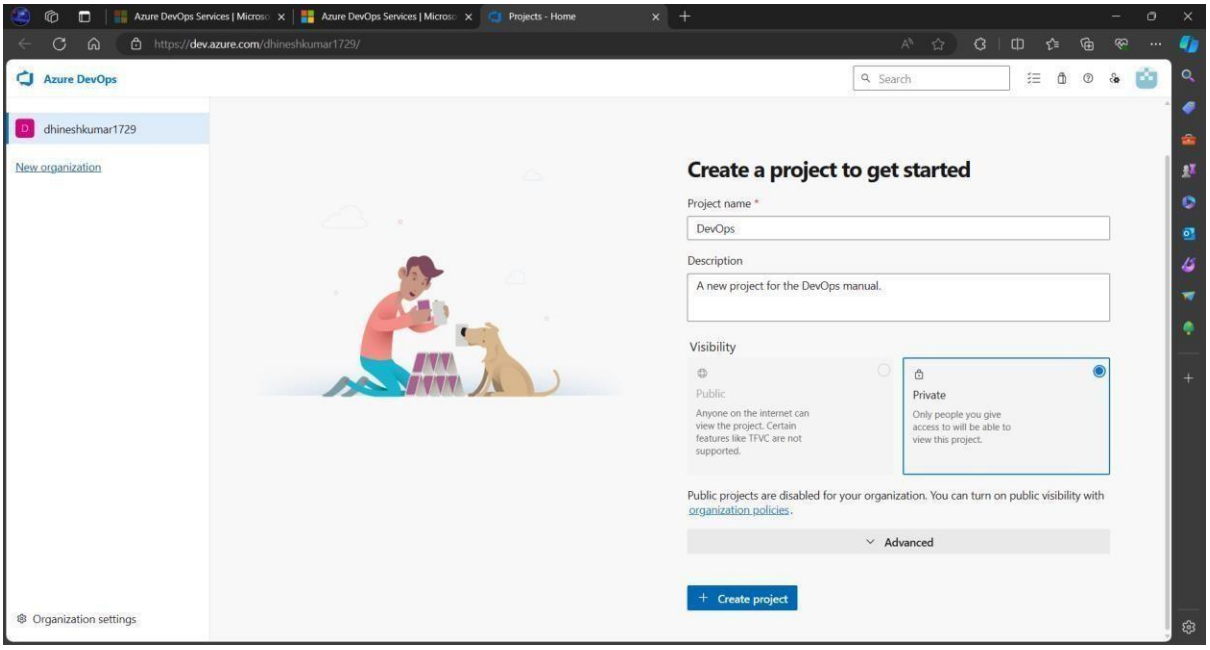
1. Go to https://azure.microsoft.com/en-in/products/devops/ website and sign in using a GitHub account.



2. Create a new organization with a suitable name of your choice. Here dhineshkumar1729 is the organization name.



3. Create a new project named **DevOps** in private mode.



STEP 2: Installing IntelliJ IDEA

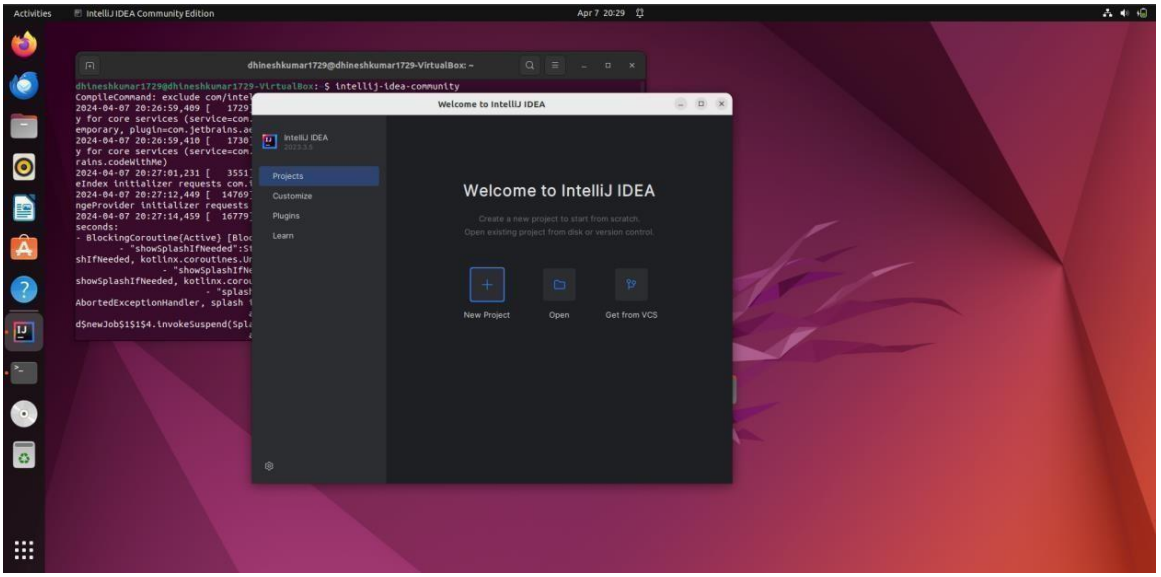
1. Download IntelliJ Idea from https://www.jetbrains.com/idea/download/?section=linux and activate a free trialfor 30 days or through the following command.

   $ sudo snap install intellij-idea-community --community

2. Open the IDE using the below command in the terminal.

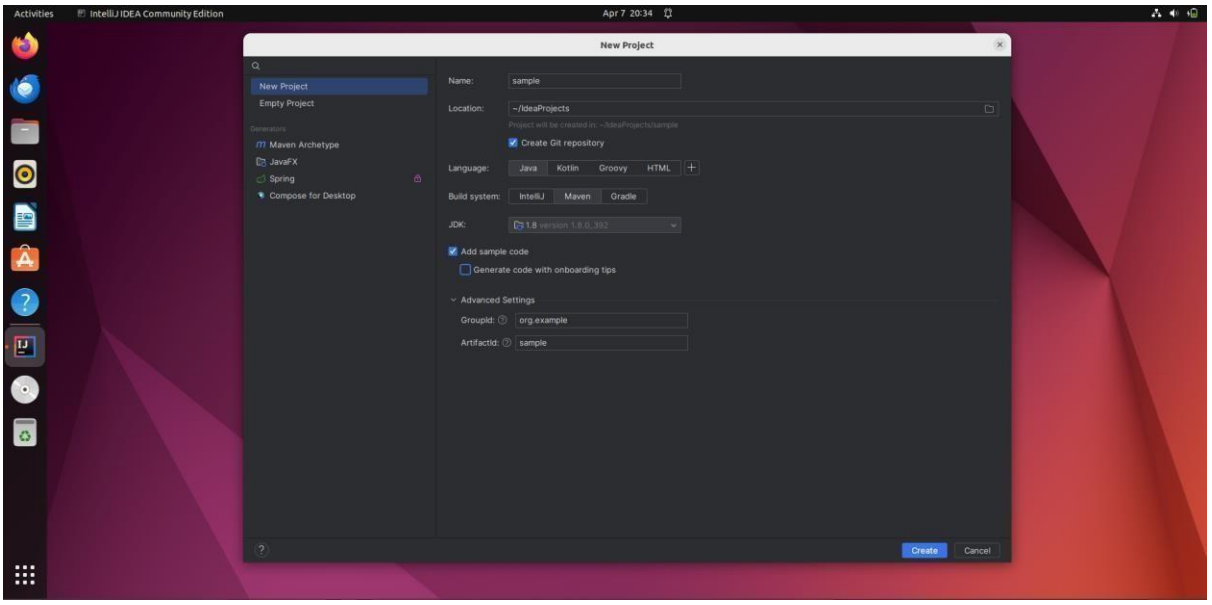
   $ intellij-idea-community

3. Java is a pre-requisite for using this IDE



STEP 3: Create a new Maven project using the IDE.

1. Create a project by clicking on the New Project option, choose an appropriate name and select the following features as mentioned in the image (Build system: Maven andLanguage: Java)



1. A sample code has been generated. Now press ALT + F12 to open a terminal insidethe IDE.

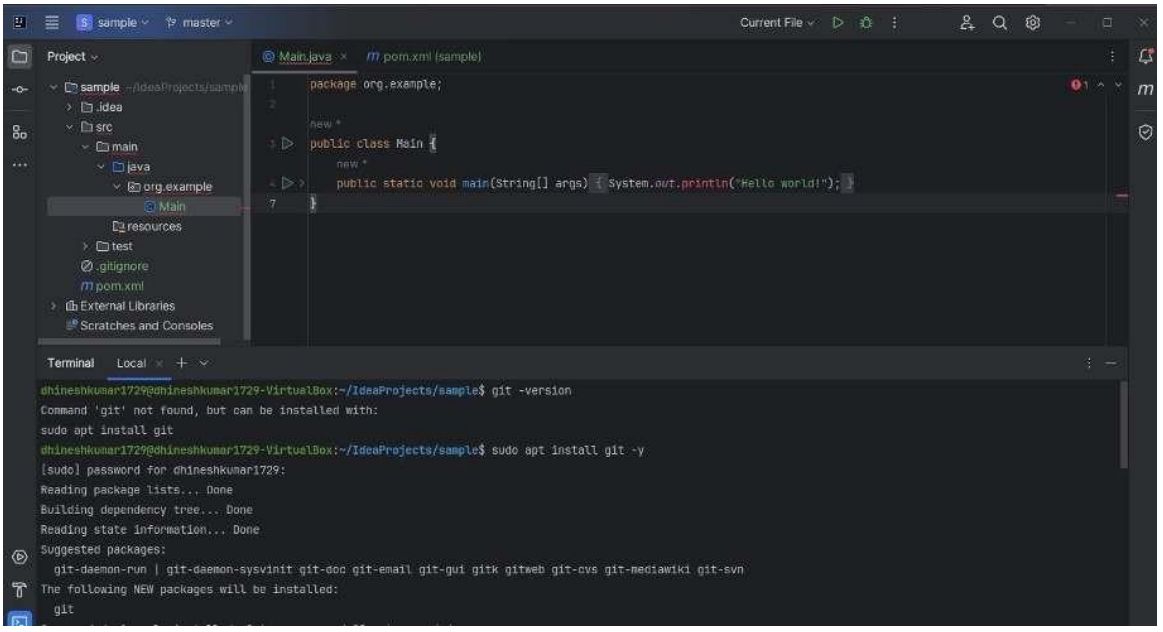
STEP 4: Install GIT and configure it

1. Check whether git is installed using the following command.

$ git -version

2. Install git using

$ sudo apt install git



3. Once git has been installed execute the following commands one by one.

$ git init

$ git add .

$ git commit -m “First Commit”

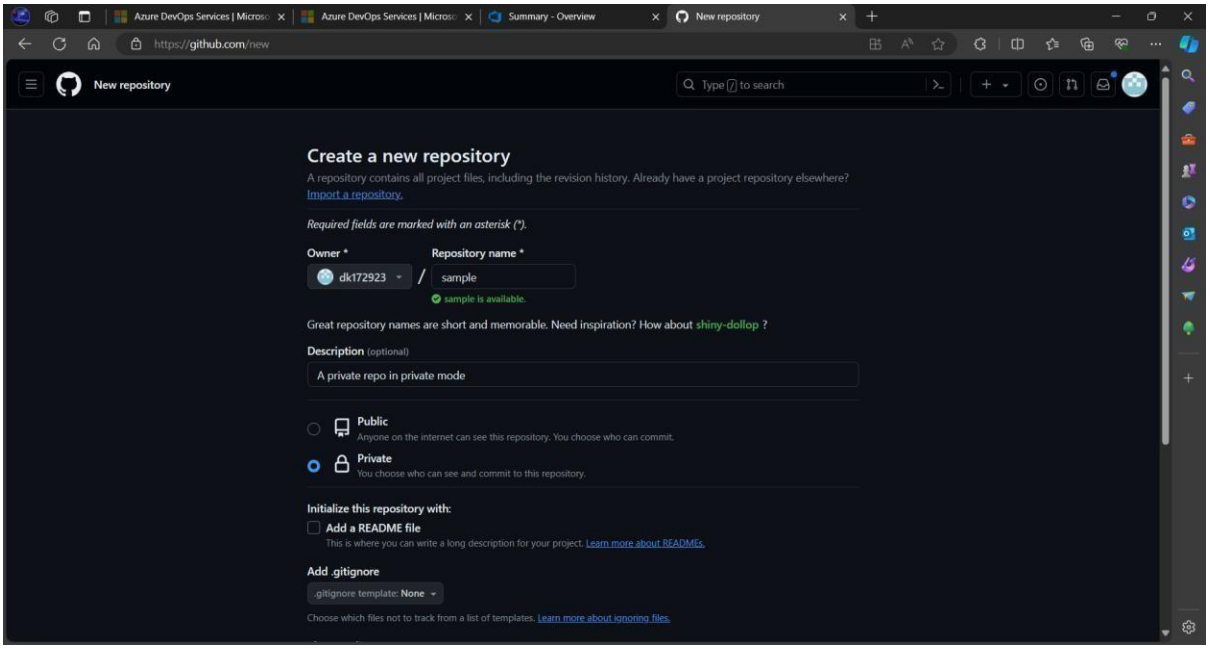
$ git branch -M main

1. Configure git using the below commands

$ git config --global user.name “Your_Name”

$ git config --global user.email “Your_Email_ID”

1. Go to https://github.com sign in with your account and create a new private repositorywith the same name as your maven project.



1. Copy the **SSH** URL for the created repository.

2. Now go back to IDE’s terminal and execute the following commands

$ ssh-keygen -t rsa -b 4096 -C dhineshkumar24murugan007@gmail.com

(press ENTER for all questions)

$ cat ~/.ssh/id_rsa.pub (copy the printed SSH key)

1. Now go to the **SSH and GPG** section in the GitHub settings option.

2. Click on the **New SSH key** button.

3. Choose a suitable name and paste the SSH key into the provided space.

4. Now get back to the IDE’s terminal and type the following command

$ git remote set-url origin git@github.com:dk172923/sample.git

(Here, git@github.com:dk172923/sample.git is the remote URL to use SSH

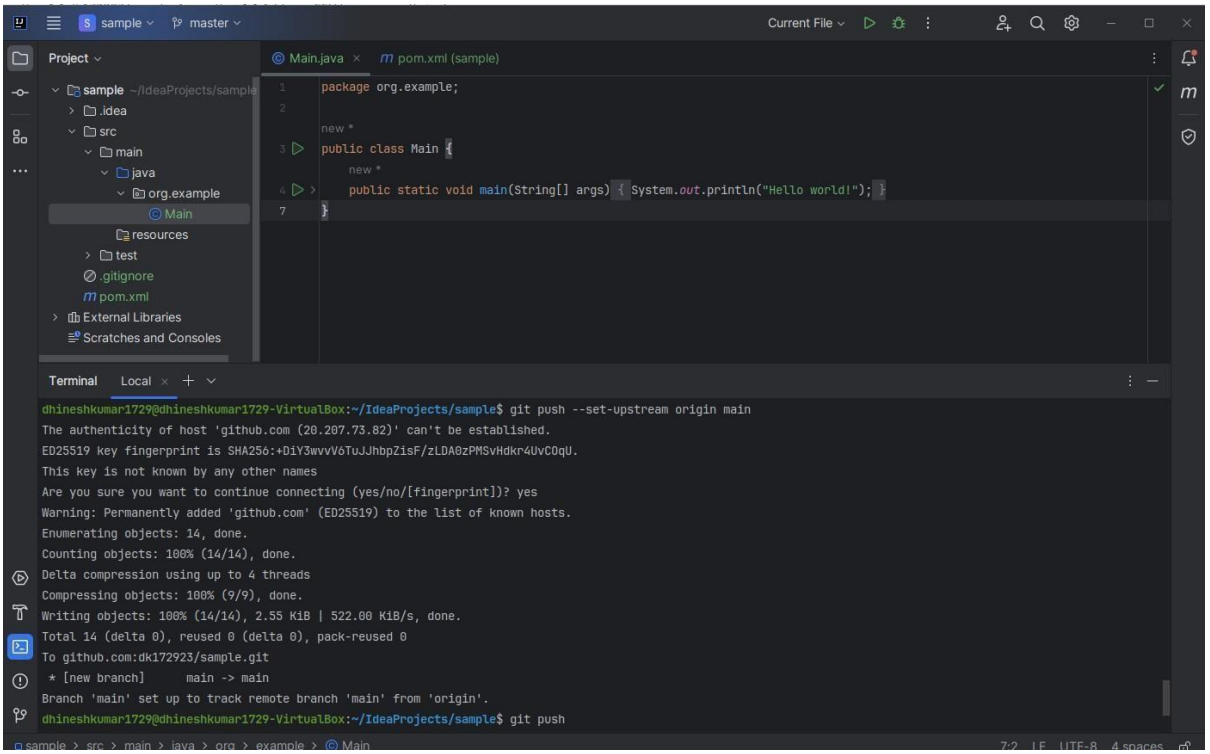
dk172923 – GitHub ID

sample.git – repository name)

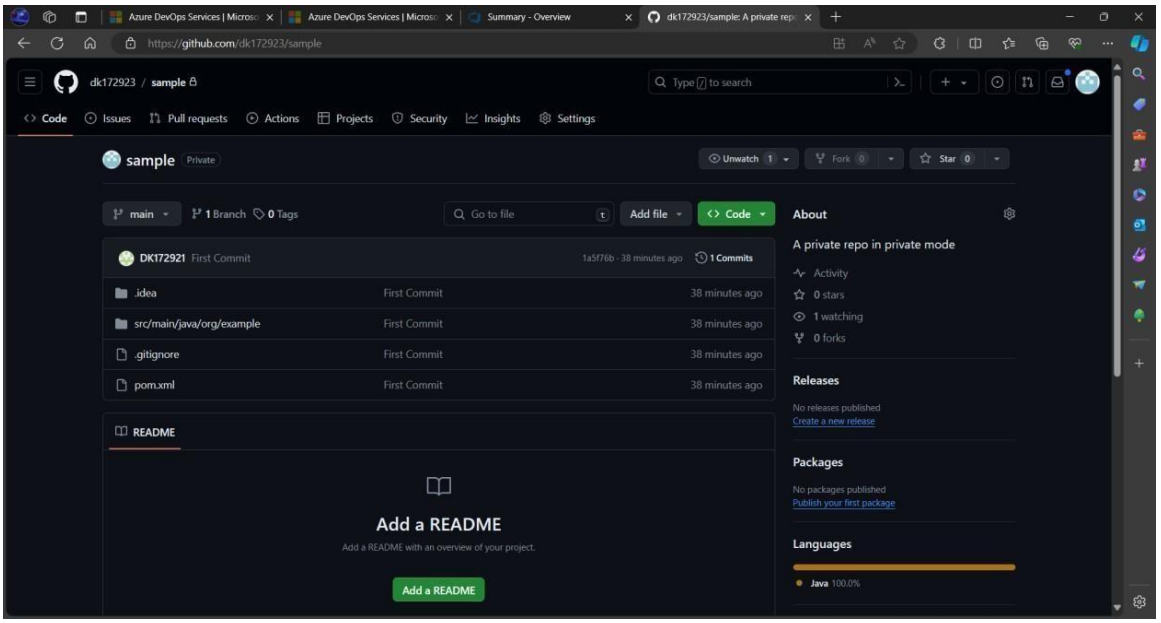
1. Now finally run the push commands to push the code to the remote repository fromthe local repository.

$ git push --set-upstream origin main (Type **YES** for prompted question)

$ git push

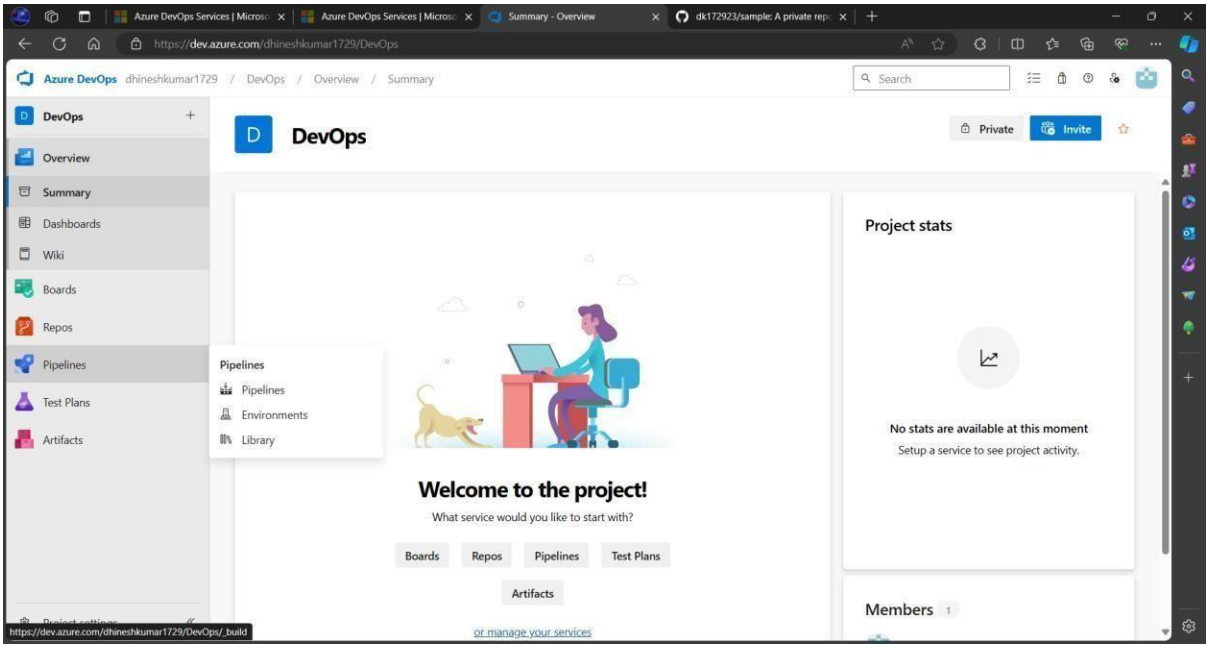


The code has been successfully pushed to the remote repository.

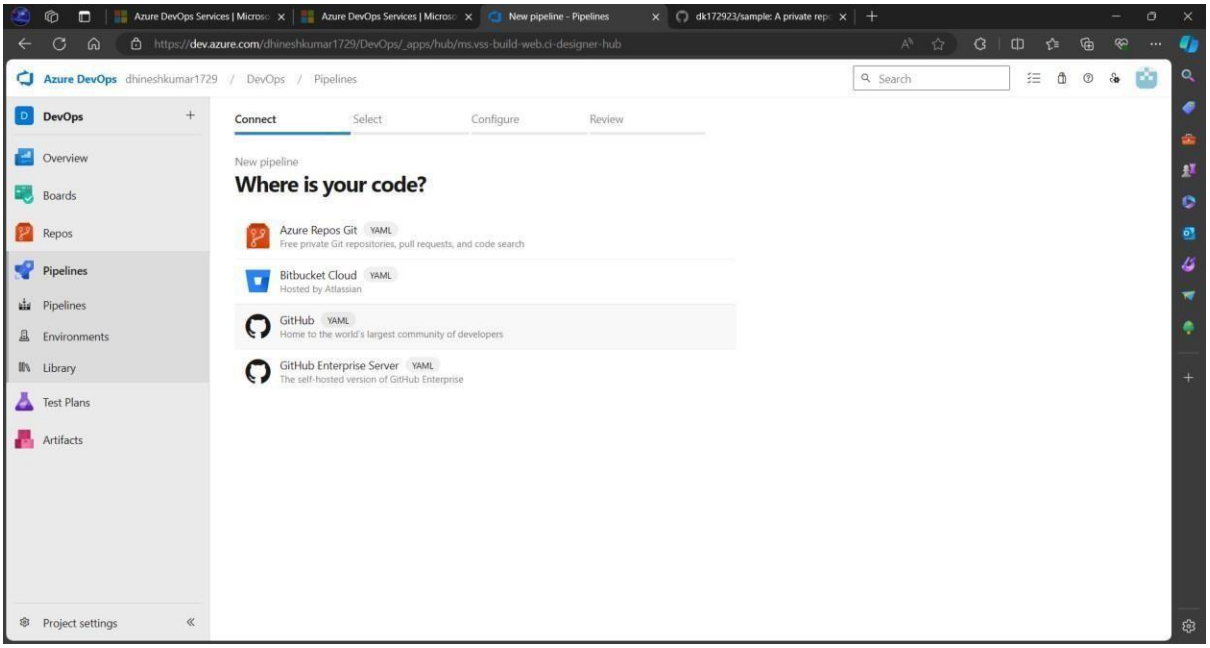


STEP 5: Create a pipeline for the created maven repository.

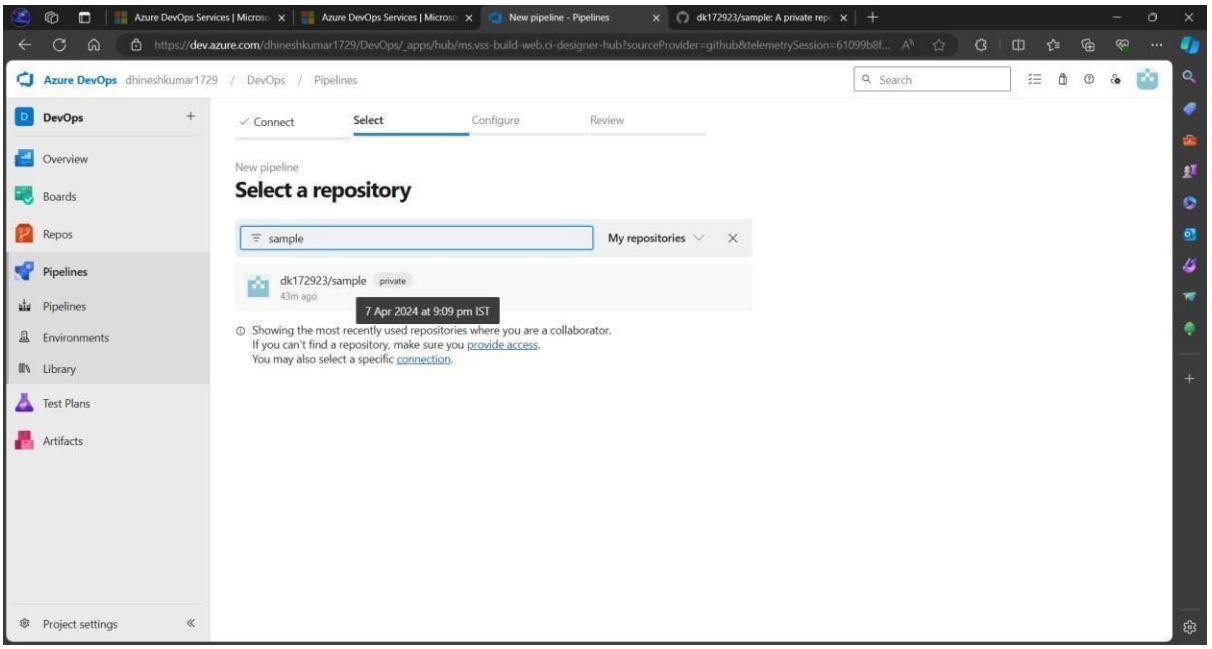
1. Click on Pipelines on the left pane of the Azure DevOps website.



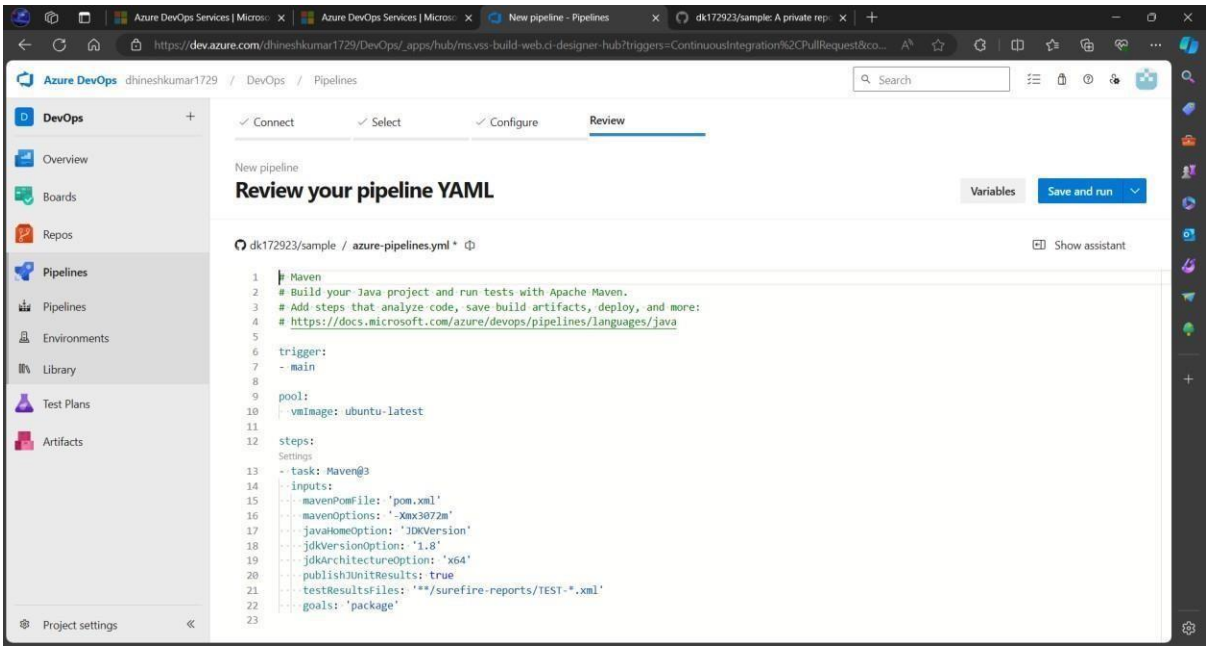
1. Click on the **Create Pipeline** button.
2. Choose the GitHub YAML option.



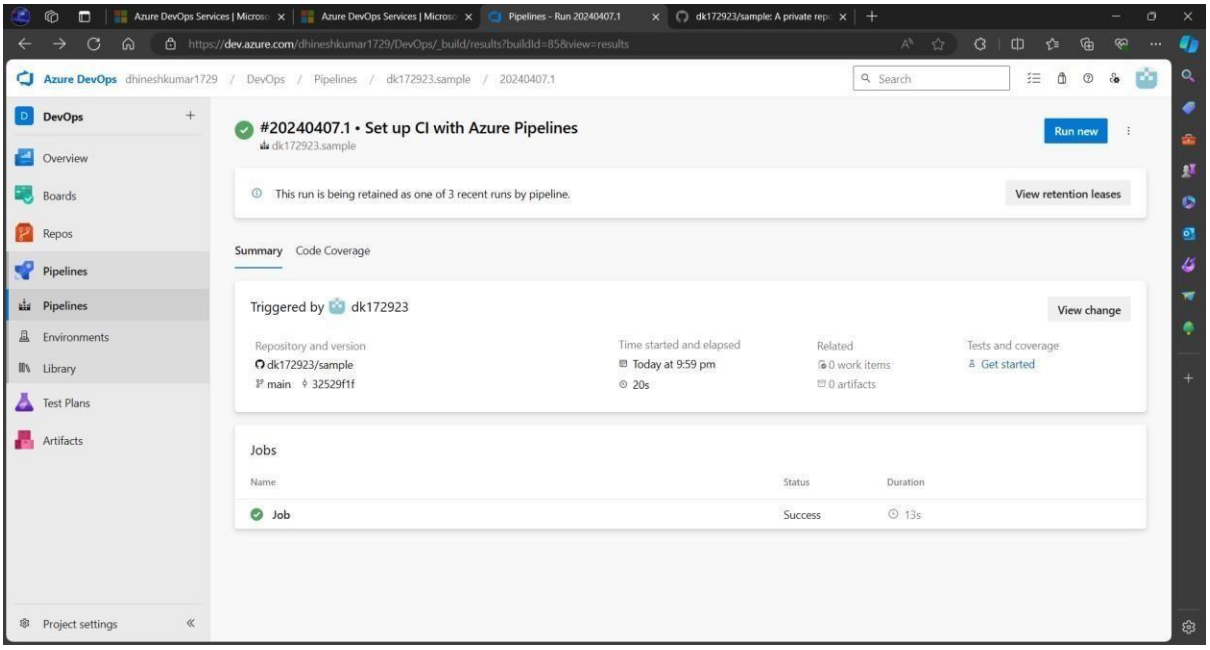
1. Select the created repository “sample” from the available repositories.
2. Give permissions if prompted to do so and sign in to your GitHub account if needed.



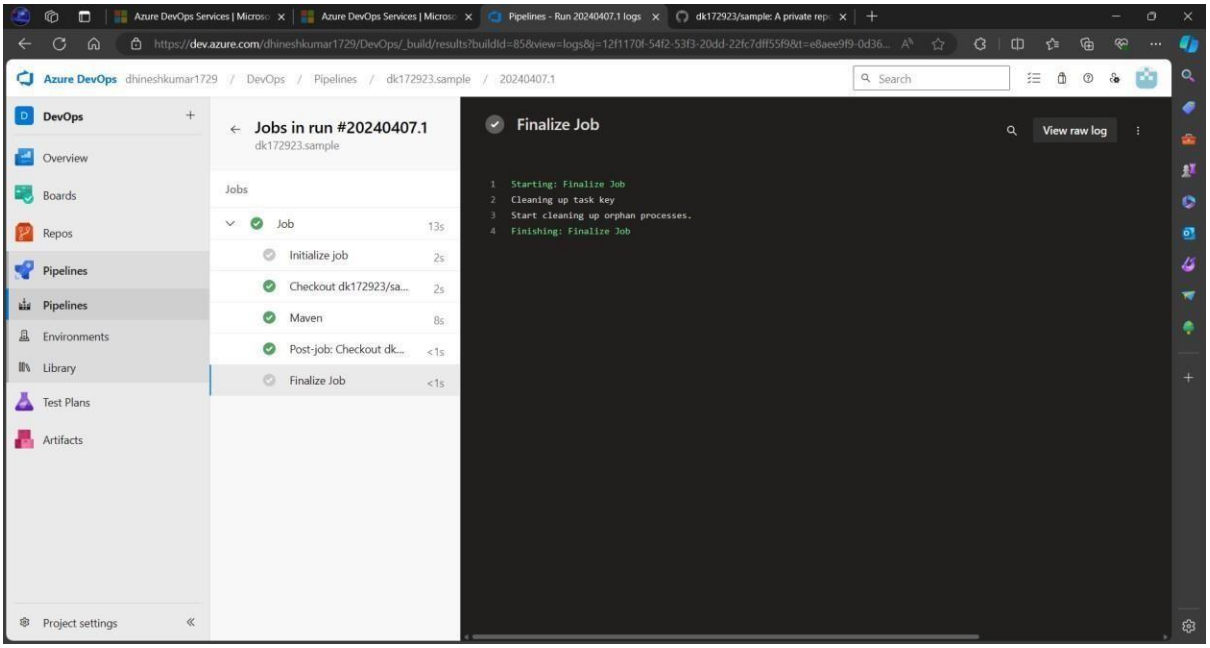
3. Choose Maven Pipeline from the given options.
4. A YAML file is automatically created based on the configuration details found in the pom.xml file in the repository.



1. Press the **Save and Run** button.
2. Create a commit message and press the run button again.
3. Click on the created Job from the Jobs table.



A successful job completion would be like this.



RESULT

Thus, a maven build pipeline has been created using Azure.

Ex. No: 2 **RUN REGRESSION TESTS USING MAVEN BUILD**

Date: **PIPELINE IN AZURE**

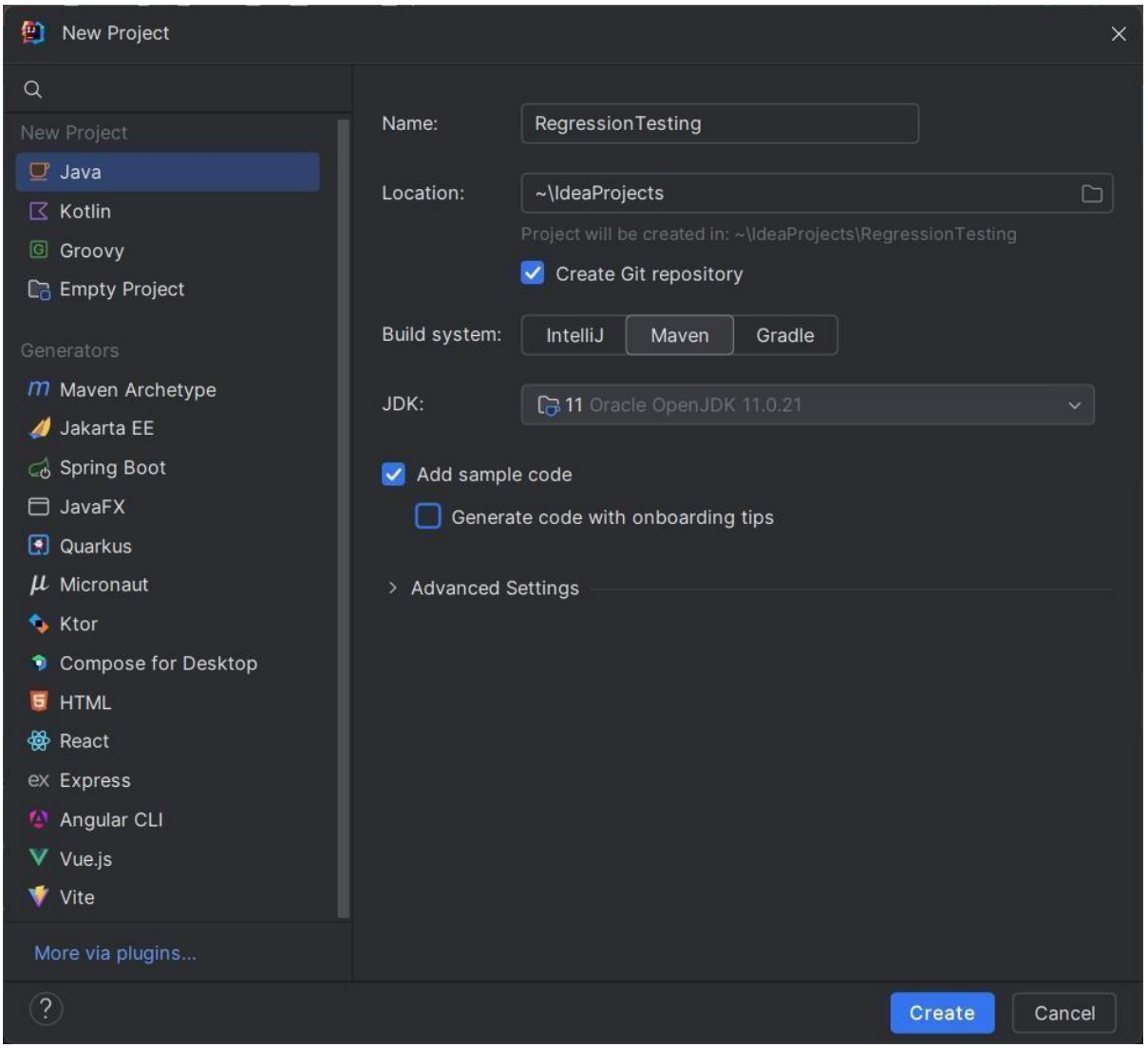
AIM

To run regression tests using the Maven build pipeline in Azure.

PROCEDURE

STEP 1: Create a new maven project in IntelliJ IDEA IDE.

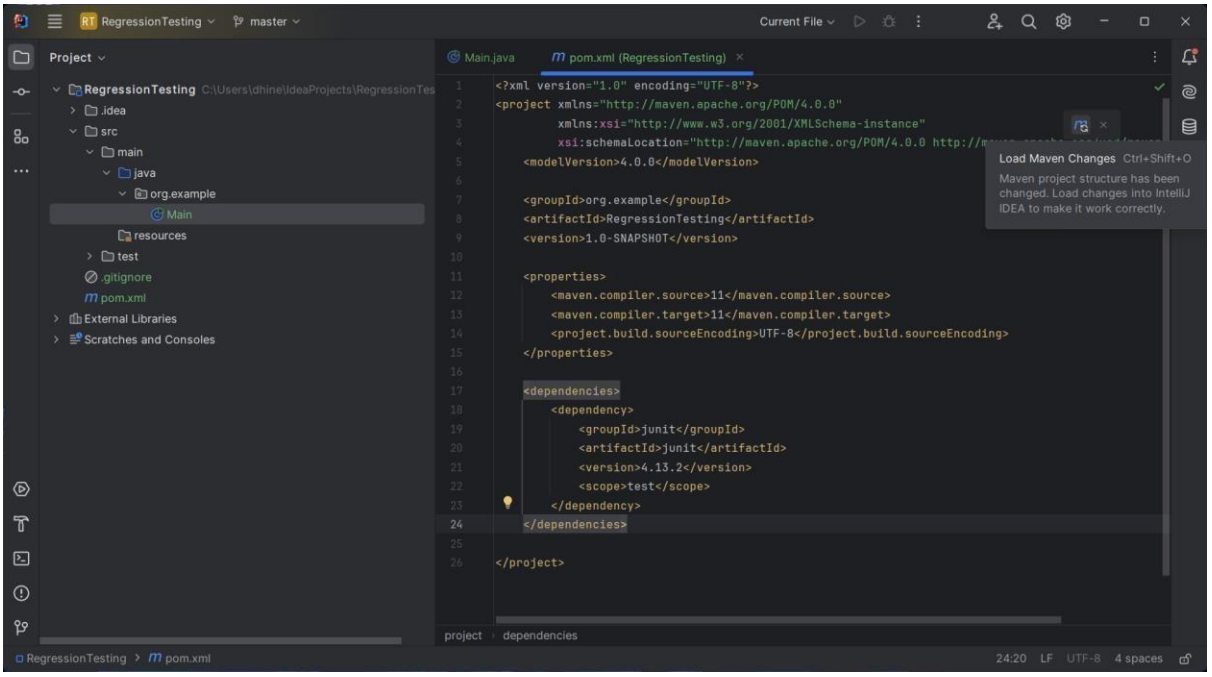
1. Create a project with **RegressionTesting** as its name.
2. Choose the **Maven** build system and available JDK version.
3. Select the **Create Git Repository** option.



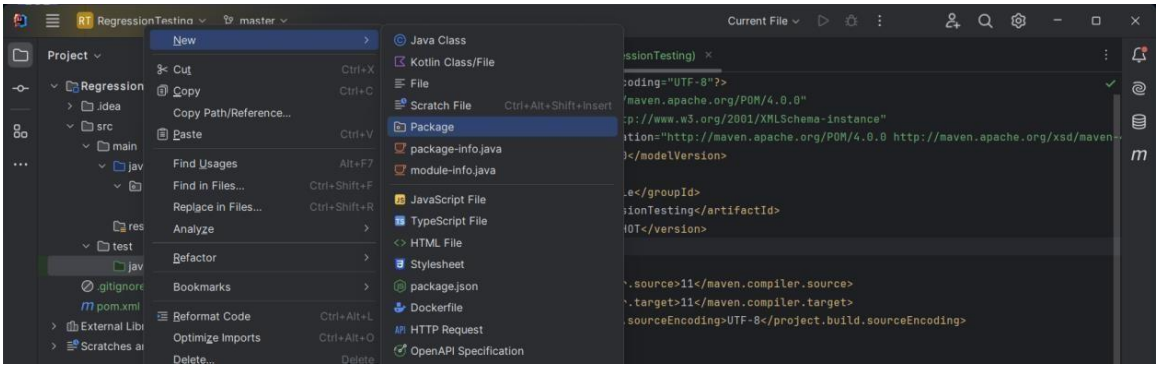
4. Wait for some time until the indexes of the project have been updated.
5. Add the following snippet of code into **pom.xml**

```
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.13.2</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

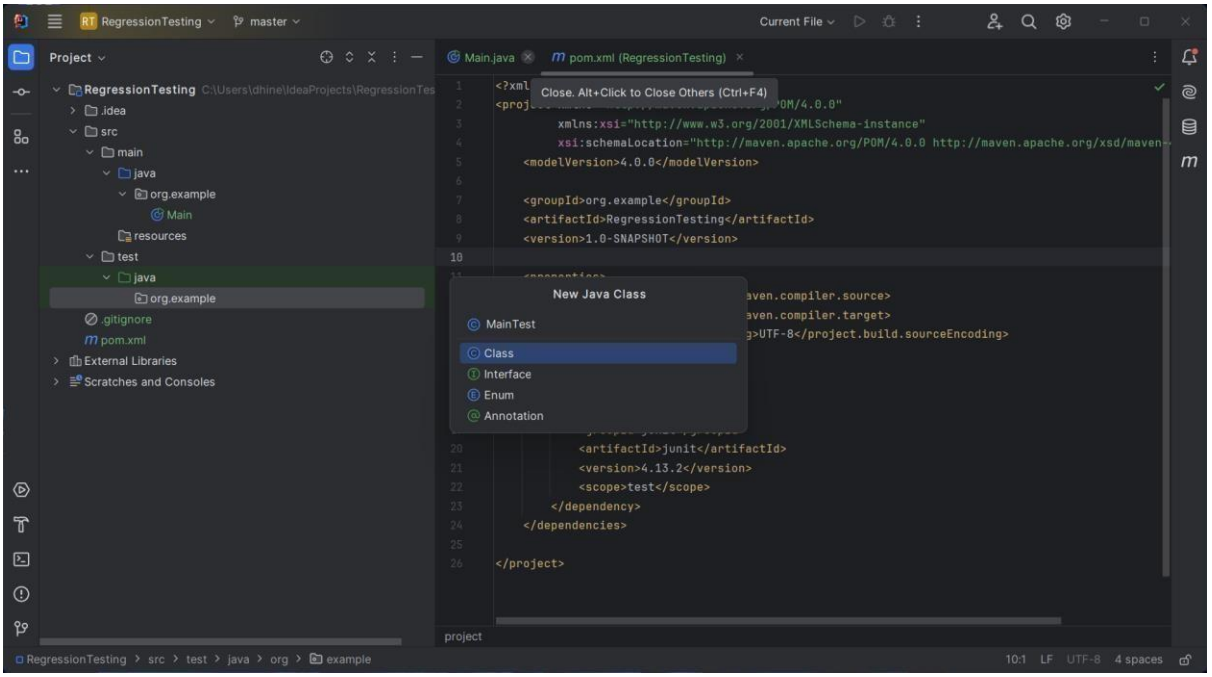
6. Once the changes have been made click on the referred button to update the changes made to pom.xml or for simplicity just press **CTRL + SHIFT + O**



7. Now, inside the java folder on the **Test** directory create a package named **org.example (**Right click on java folder inside the Test directory choose New and select package**)**



8. Right-click on the package and create a new class named **MainTest**

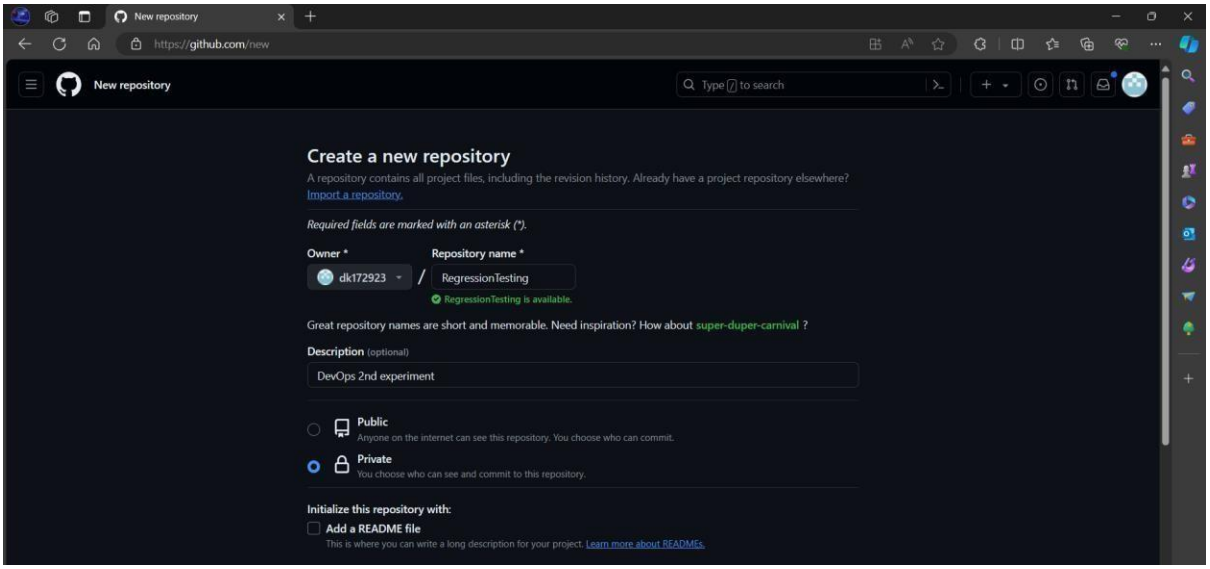


9. Paste the following into the MainTest.java file

```
package org.example;
import org.junit.Test;
import java.io.ByteArrayOutputStream;
import java.io.PrintStream;
import static org.junit.Assert.assertEquals;
public class MainTest {
    @Test
    public void testMainMethod() {
        // Redirect standard output to capture console output
        ByteArrayOutputStream outContent = new ByteArrayOutputStream();
        System.setOut(new PrintStream(outContent));

        // Call the main method
        Main.main(new String[]{});

        // Check if "Hello and welcome!" was printed
        assertEquals("Hello and welcome!", outContent.toString().trim());

        // Reset standard output
        System.setOut(System.out);
    }
}
```

STEP 2: Create a GitHub repository

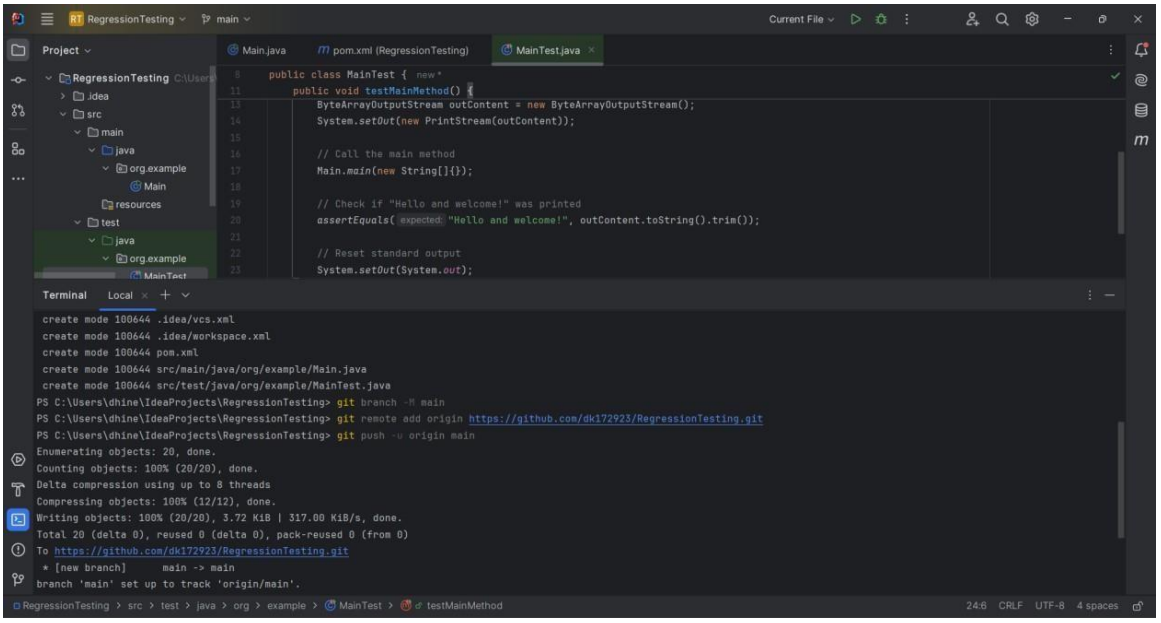
1. Create a private repository with RegressionTesting as its name.



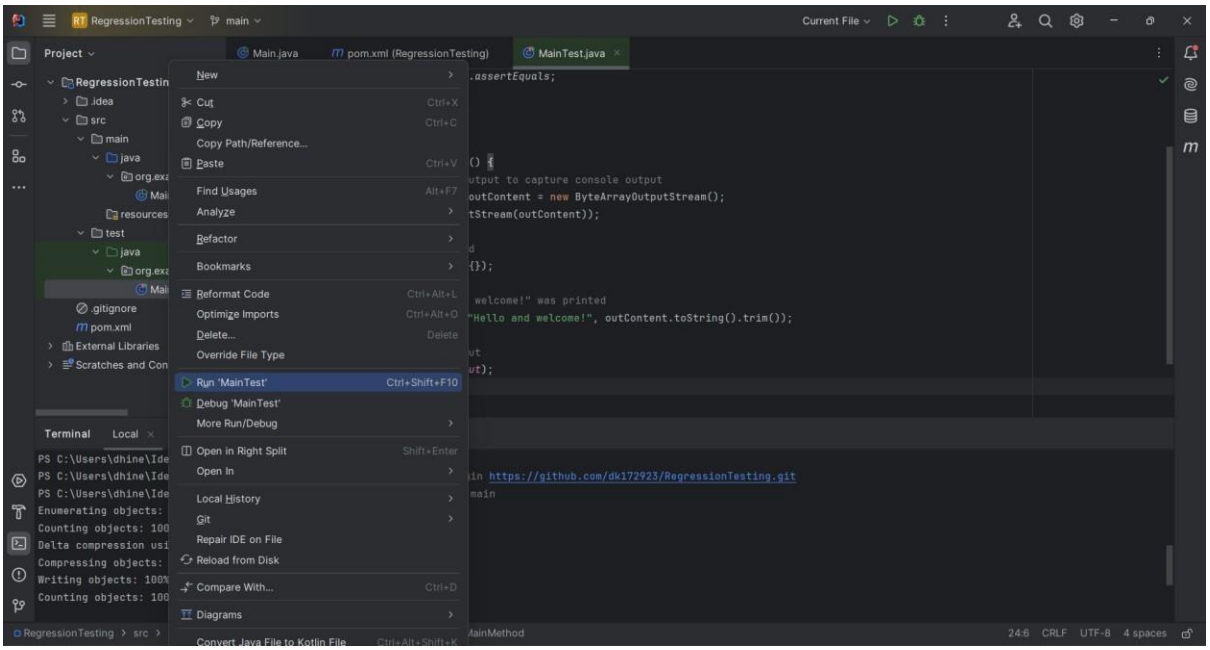
2. Execute the following commands in the terminal of IntelliJ IDEA

$ git init
$ git add .
$ git commit -m “regression”
$ git branch -M main
$ git remote add origin YOUR_REPOSITORY_LINK
$ git push -u origin main

NOTE: YOUR_REPOSITORY_LINK must contain either the HTTPS or SSH link to your respective GitHub repository.



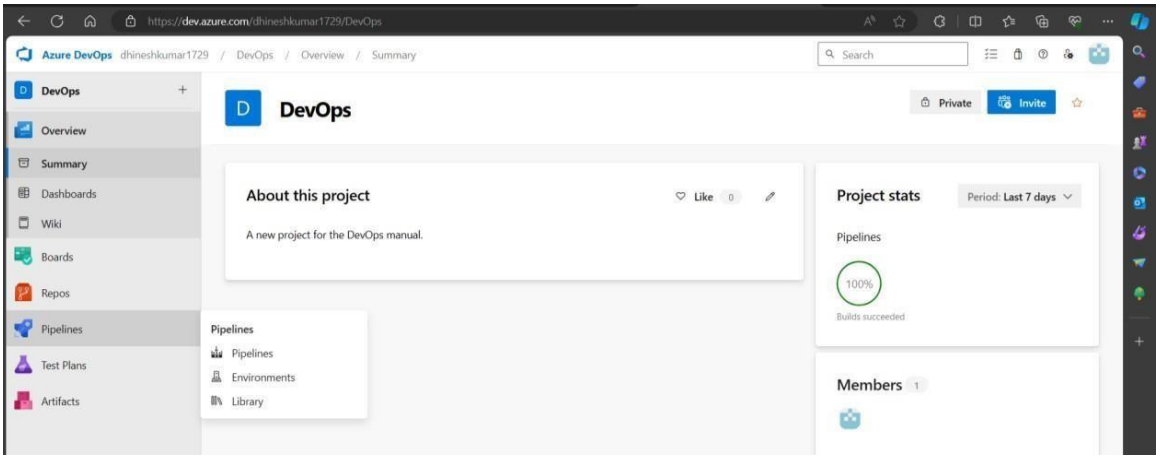
3. The MainTest class can be executed by Right-Clicking on the class from the project structure and choosing **Run 'MainTest'**



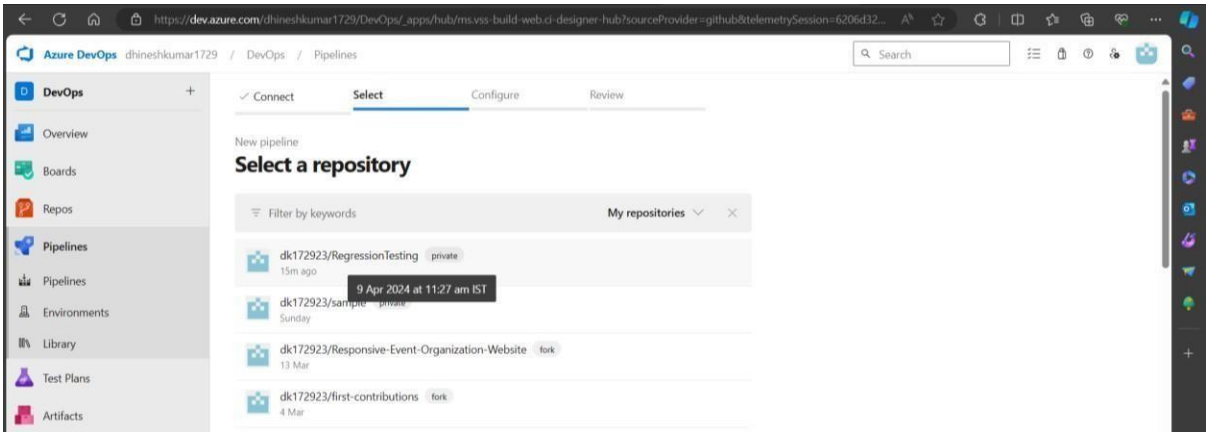
The following can be achieved on the Run tab.

STEP 3: Create an Azure Pipeline

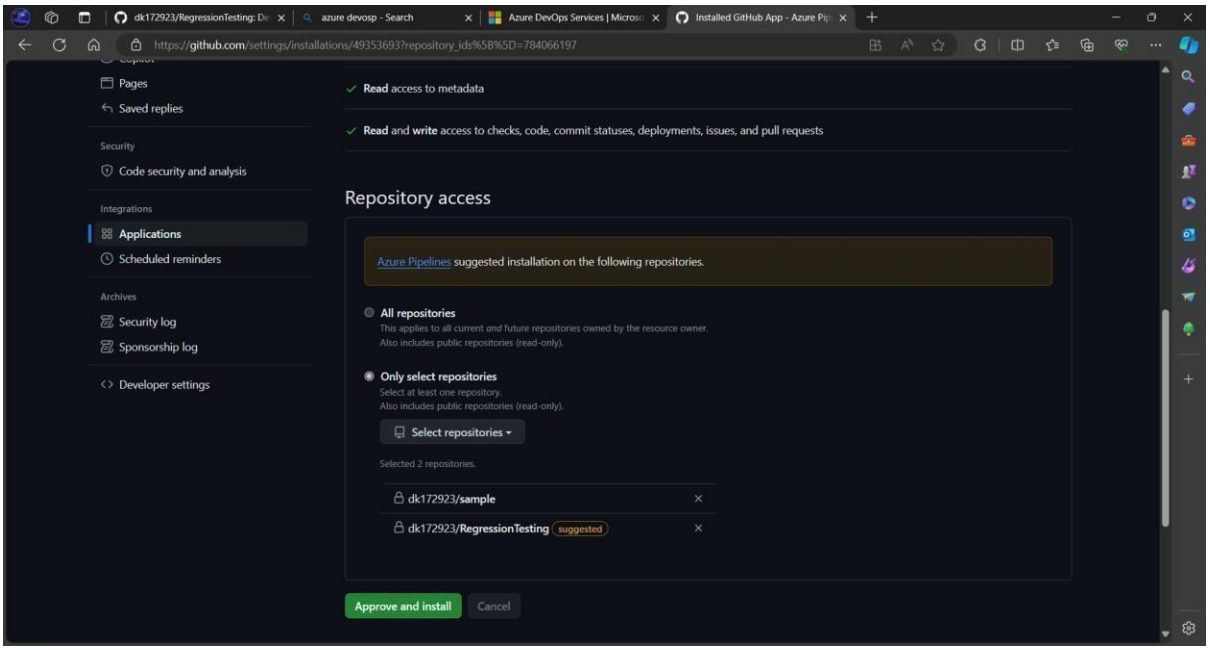
1. Open the Azure DevOps site



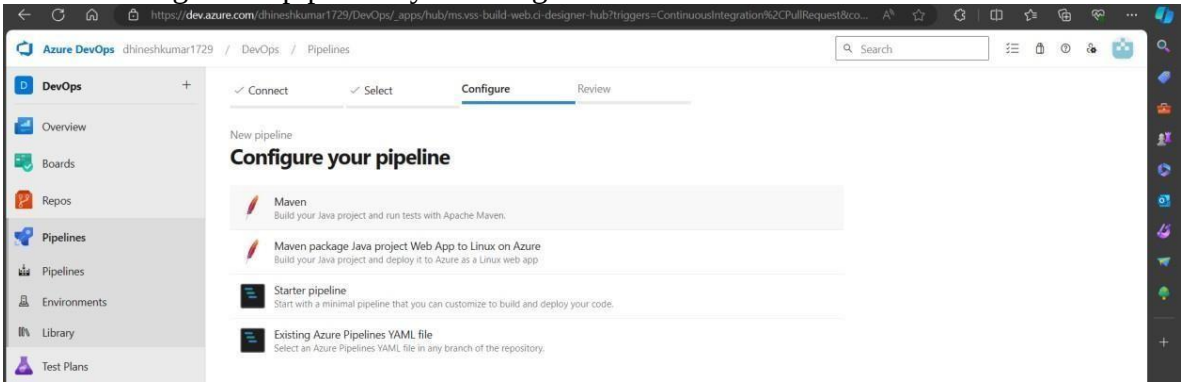
2. Click on existing project DevOps and choose Pipelines from the left pane.
3. Click on the **New Pipeline** button on the right.
4. Choose GitHub from the list.
5. Select the repository for RegressionTesting.



6. If prompted for permission then select Approve and Install on GitHub website.



7. Configure the pipeline by choosing Maven

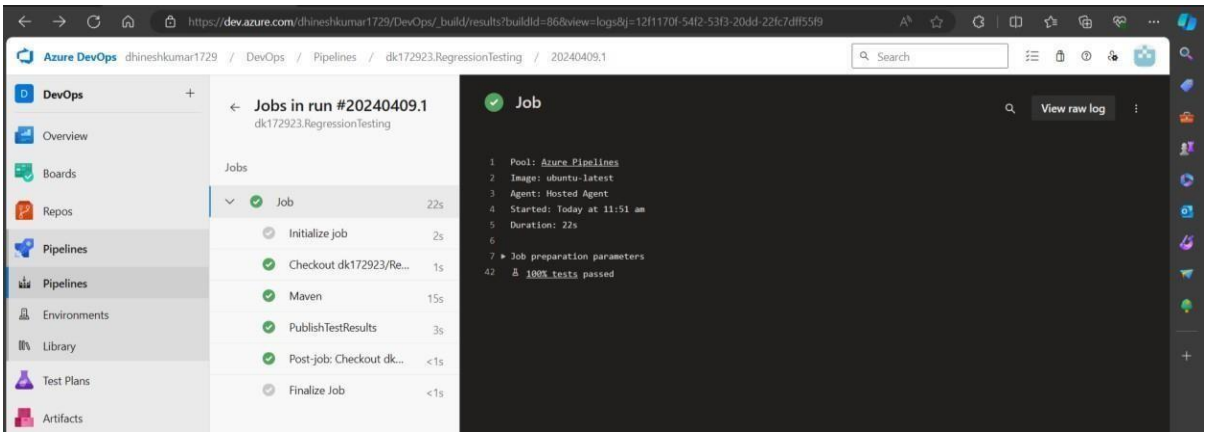


8. On the next page an azure-pipeline.yml file is generated automatically. Replace this file with the following code

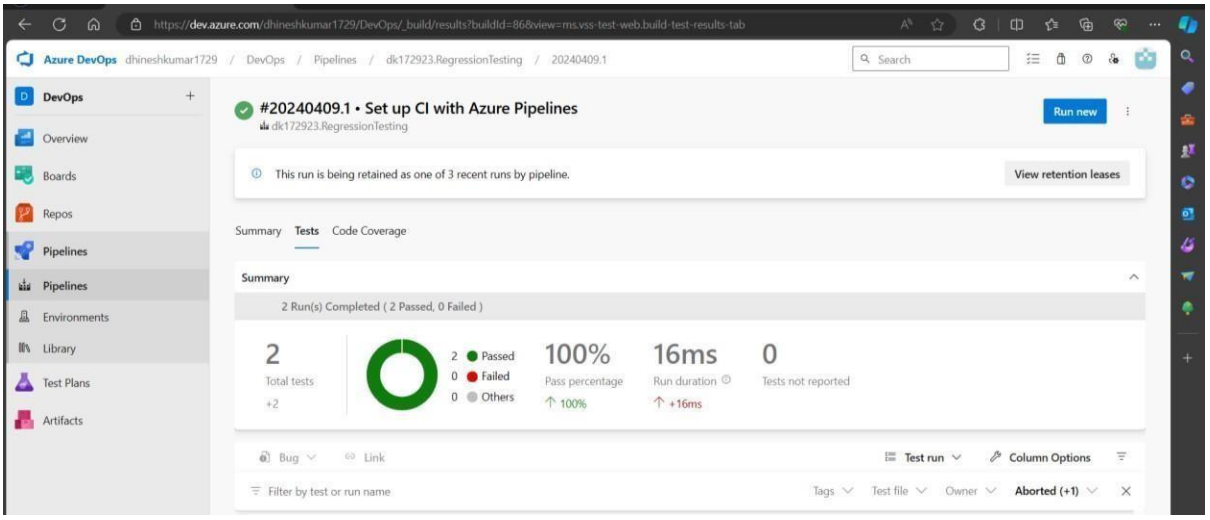
```
# Maven

# Build your Java project and run tests with Apache Maven.

# Add steps that analyze code, save build artifacts, deploy, and more:
# https://docs.microsoft.com/azure/devops/pipelines/languages/java


trigger:

- main


pool:

  vmImage: 'ubuntu-latest'


steps:

- task: Maven@3
  inputs:
    mavenPomFile: 'pom.xml'
    mavenOptions: '-Xmx3072m'
    javaHomeOption: 'JDKVersion'
    jdkVersionOption: '11'
    jdkArchitectureOption: 'x64'
    goals: 'clean test'


- task: PublishTestResults@2
  inputs:
    testResultsFiles: '**/surefire-reports/TEST-*.xml'
    testRunTitle: 'JUnit Test Results'
```

9. Click on the Save and Run button to commit the changes and execute the pipeline.

10. The following results can be obtained on the output page.



11. From the test results tab it can be found that the tests passed.



RESULT

Thus, regression tests have been run using the Maven Build Pipeline in Azure.

Ex. No: 3 **INSTALL JENKINS IN CLOUD**
Date:

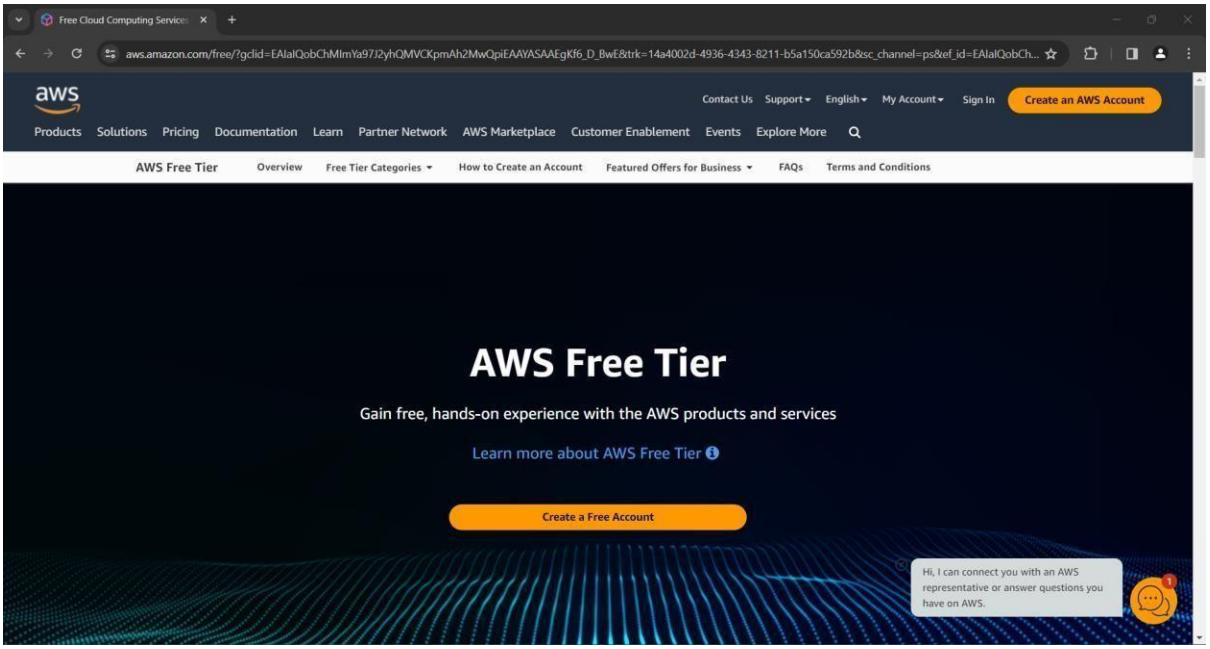
AIM

To install Jenkins in the cloud (AWS).

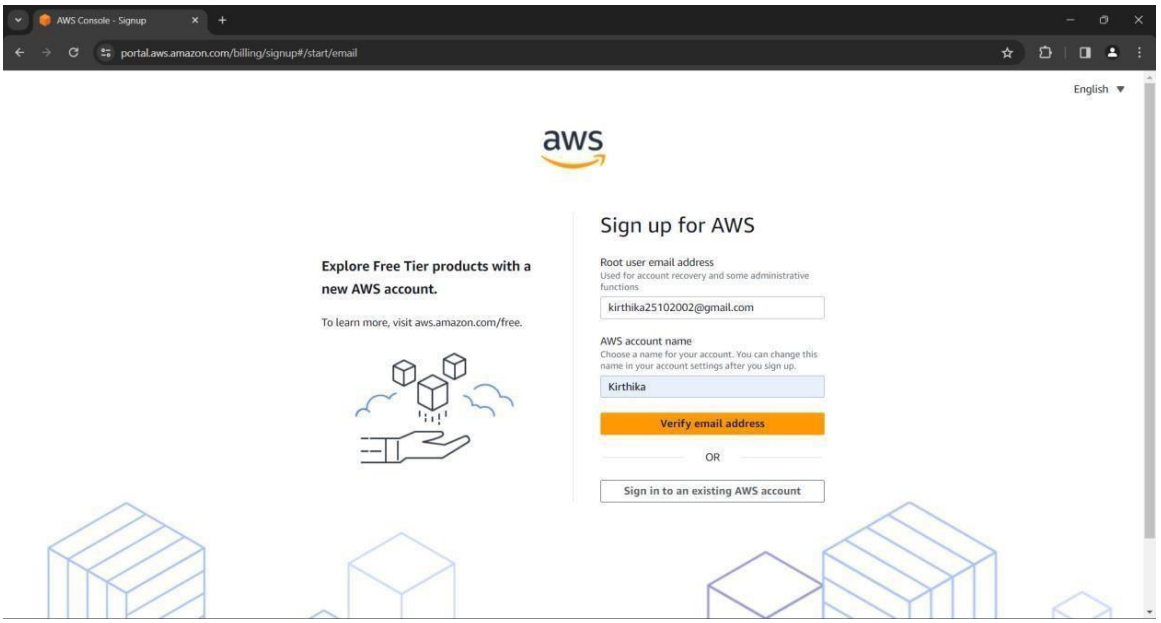
PROCEDURE

STEP 1: Create an AWS account

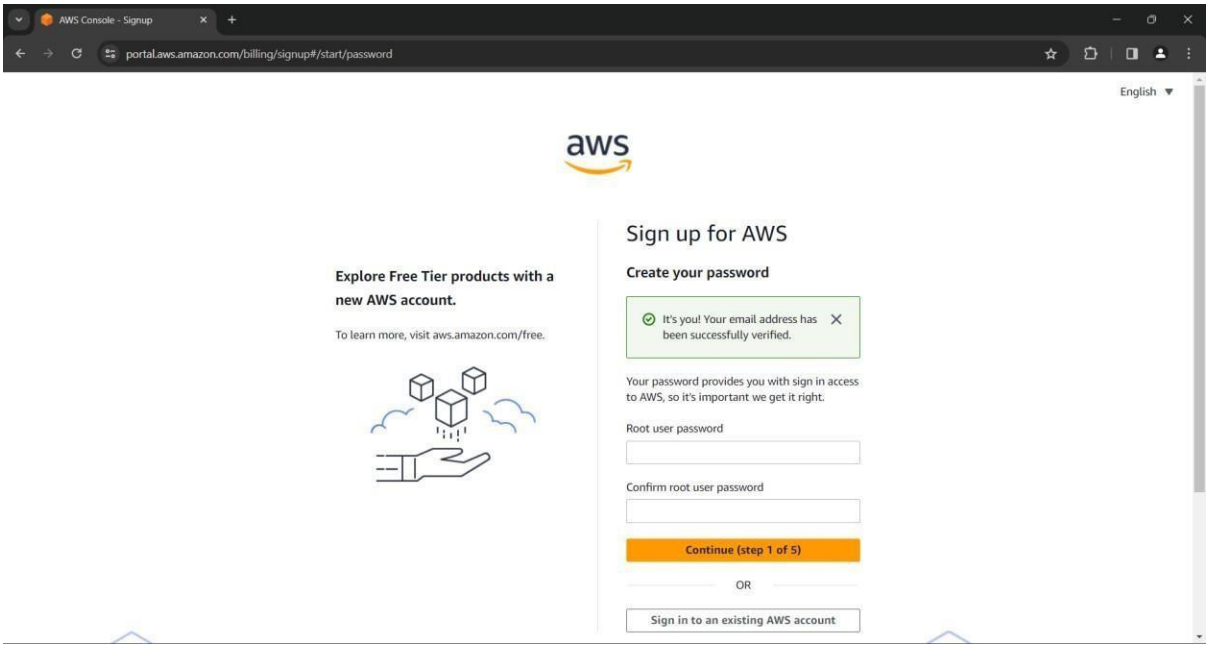
1. Go to https://aws.amazon.com/ and create an AWS Account.



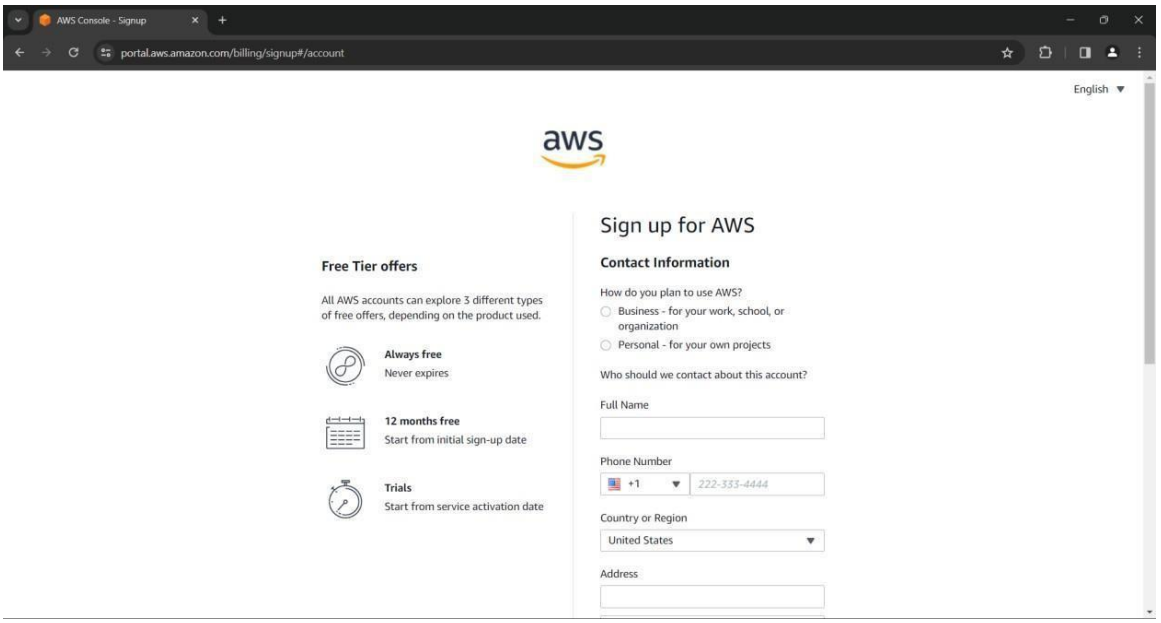
2. Provide a root email ID and username for the user.



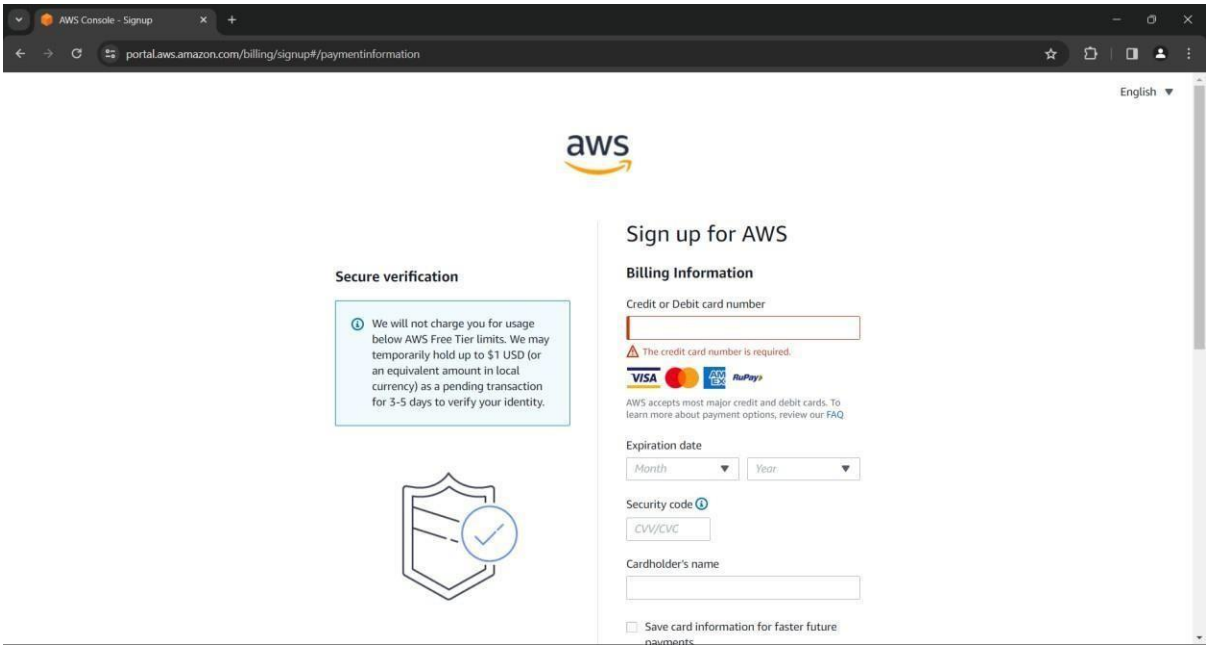
3. After Email verification set up your password.



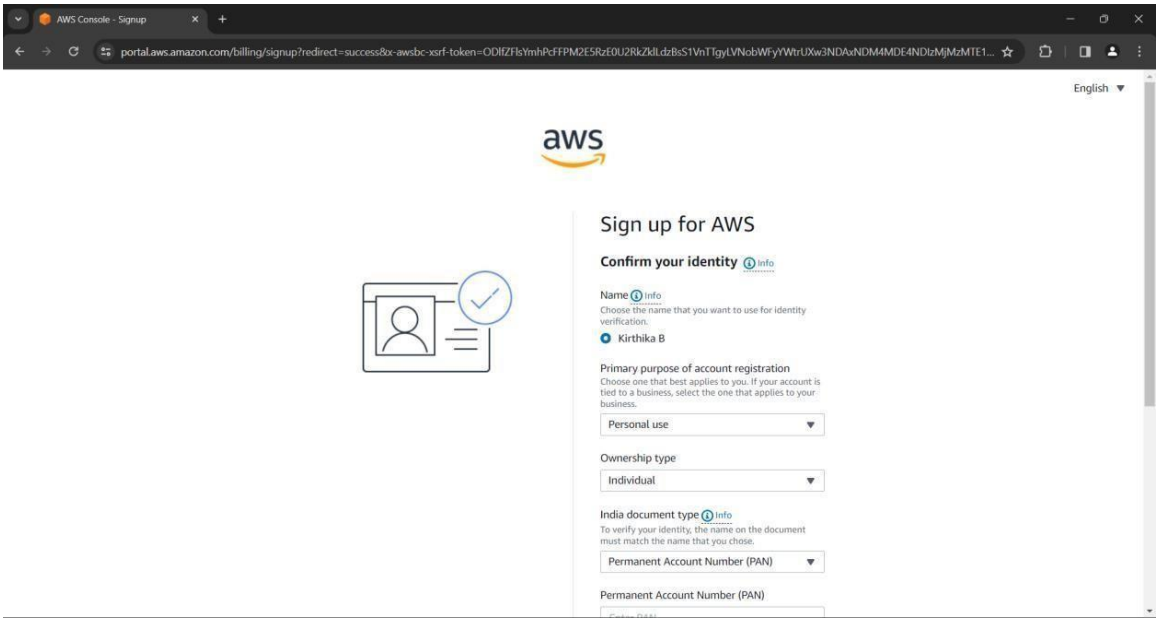
4. Complete the sign-up process by providing additional information.



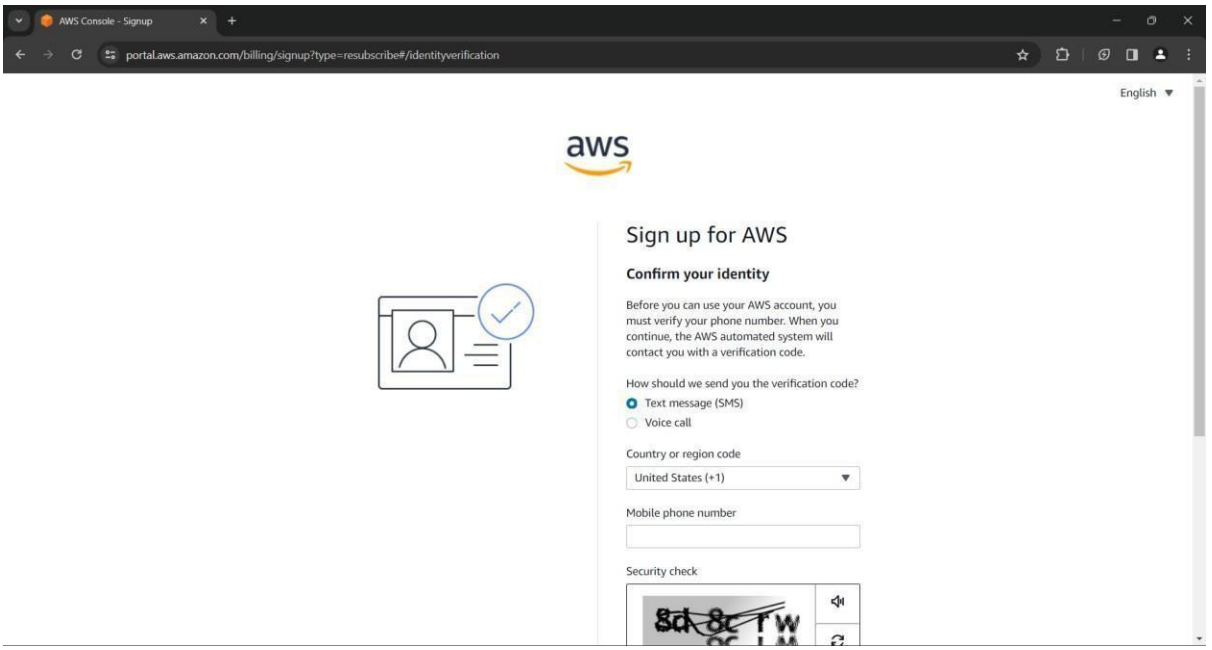
5. Provide the debit card information for the identity verification process. A charge of 2 rupees will be debited.



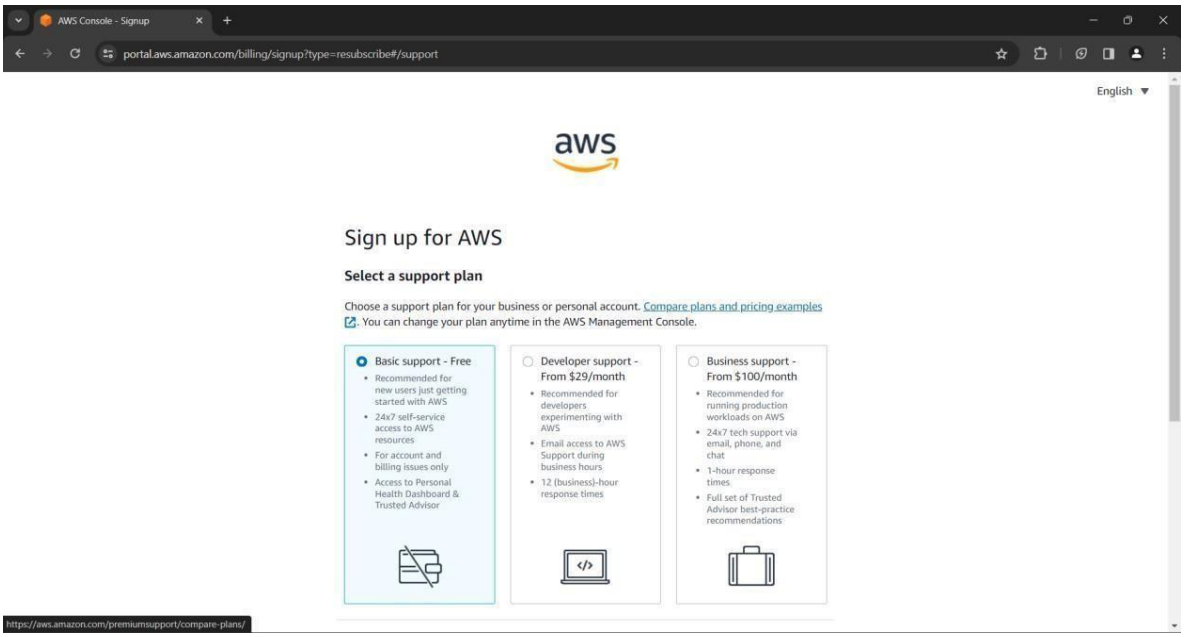
6. Confirm your identity by providing the PAN card number associated with the name of the user.



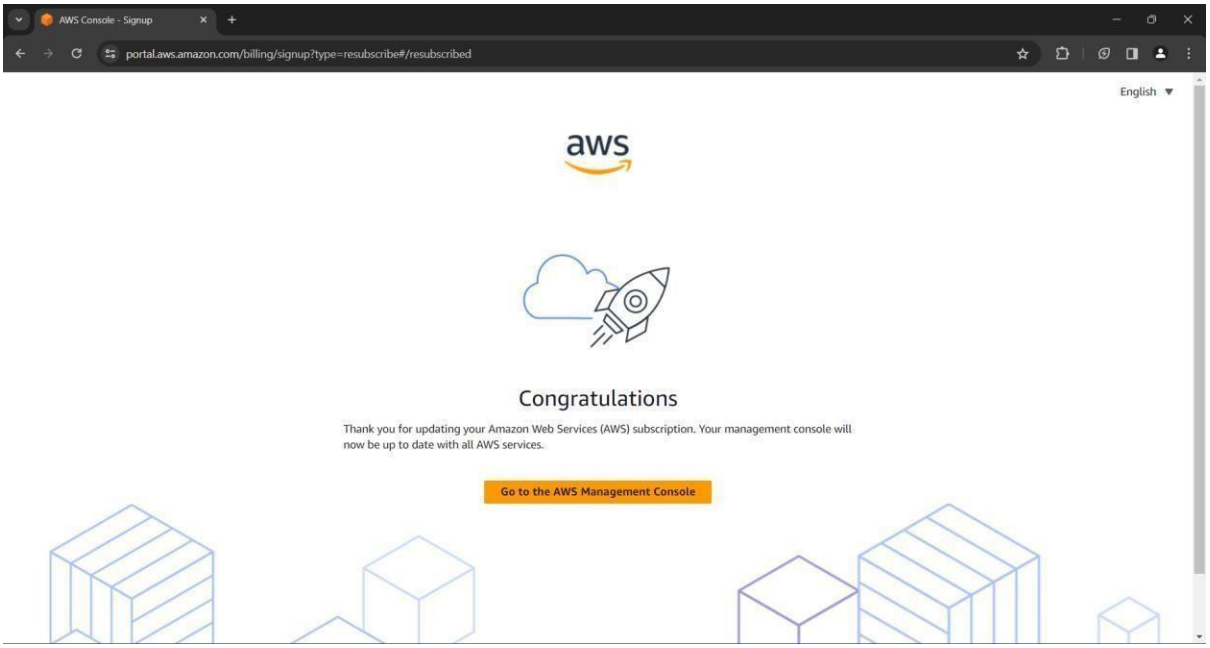
7. Verify the phone number and perform a security check.



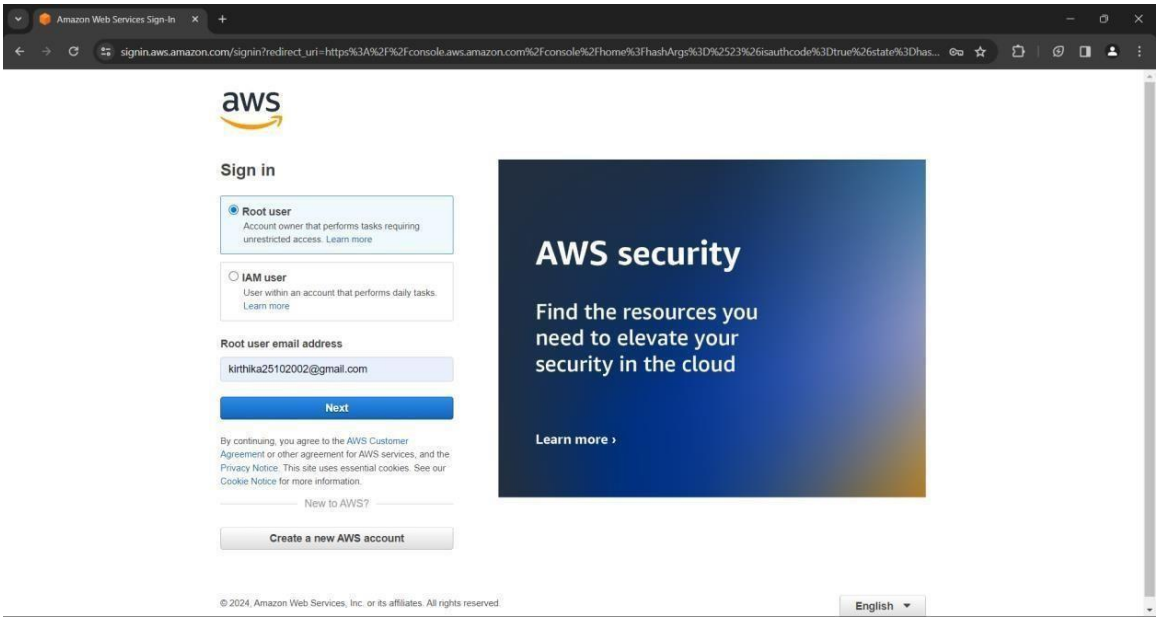
8. Select Basic Support as the plan since it is free of charge.



9. The AWS account has been successfully created and verified. Now click on **Go to the Aws Management Console** button.

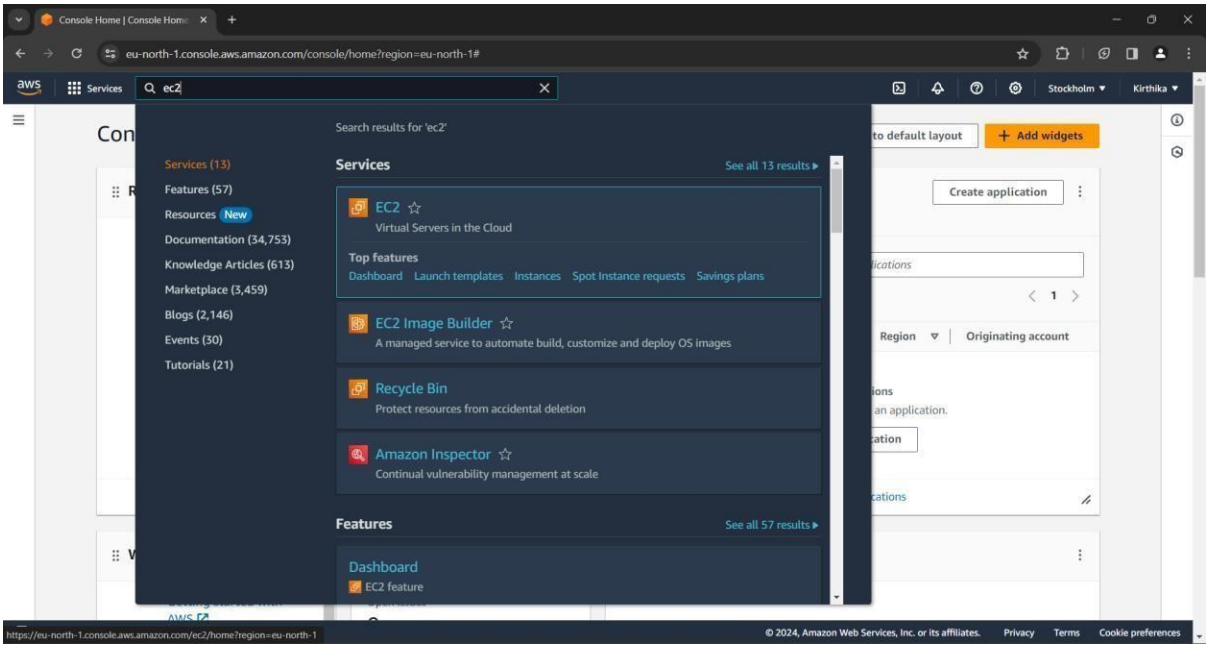


10. Sign in as a Root user by providing the appropriate email address and password.

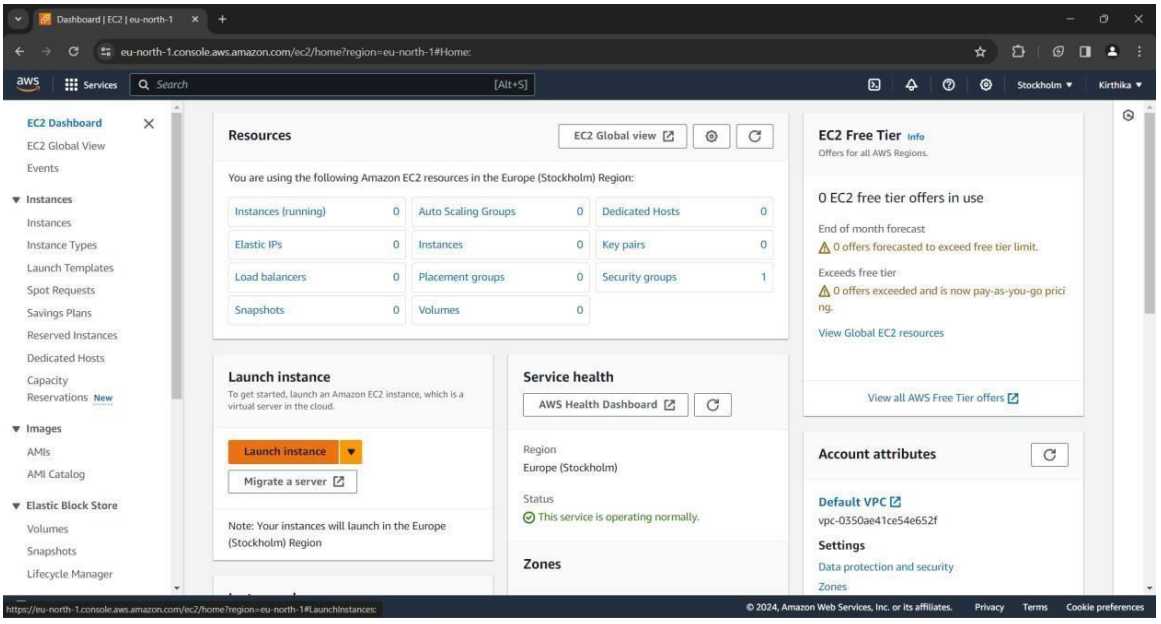


STEP 2: Create an EC2 instance.

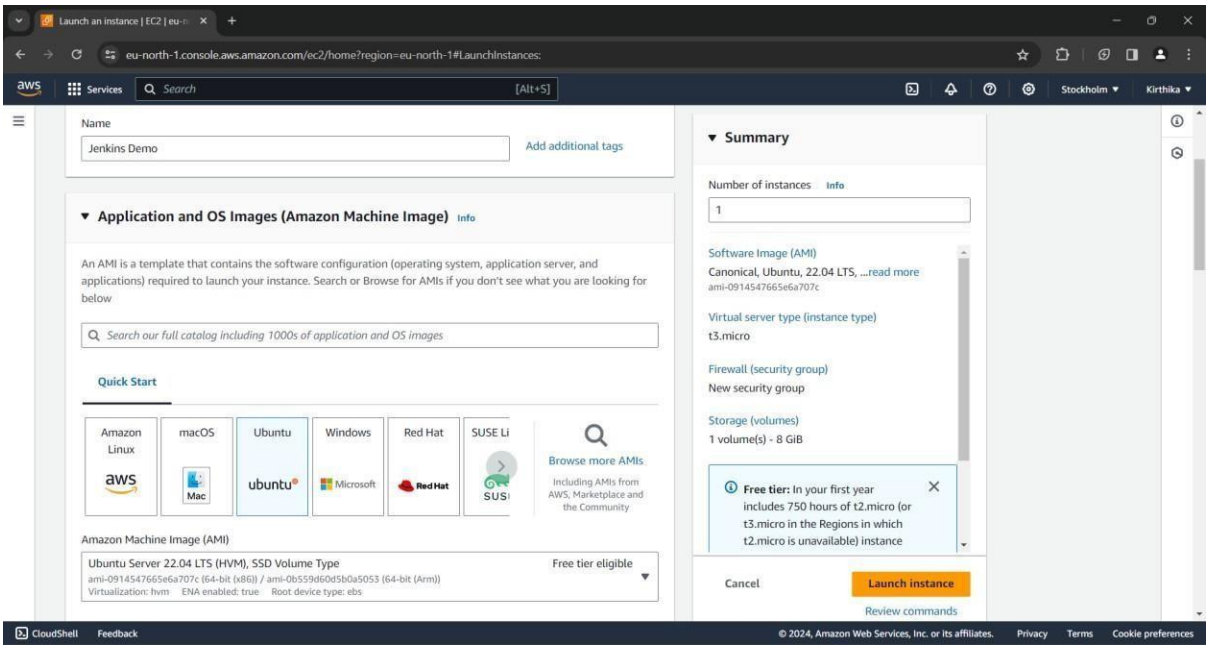
1. Search EC2 in the search bar and choose EC2 from the services tab.



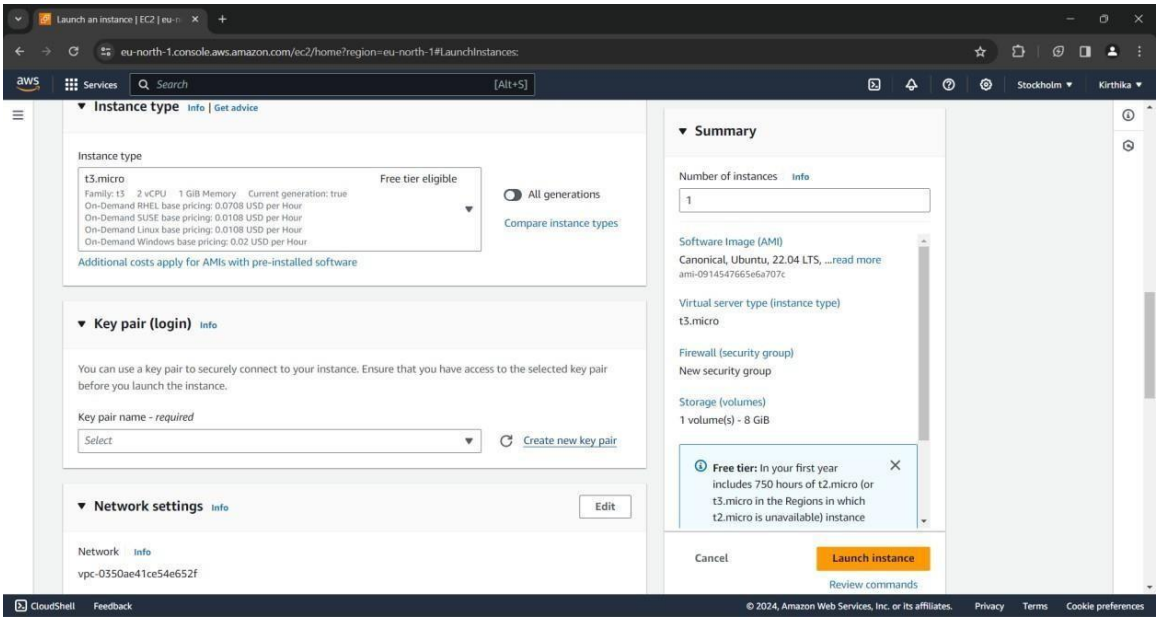
2. Click on the **Launch Instance** button.



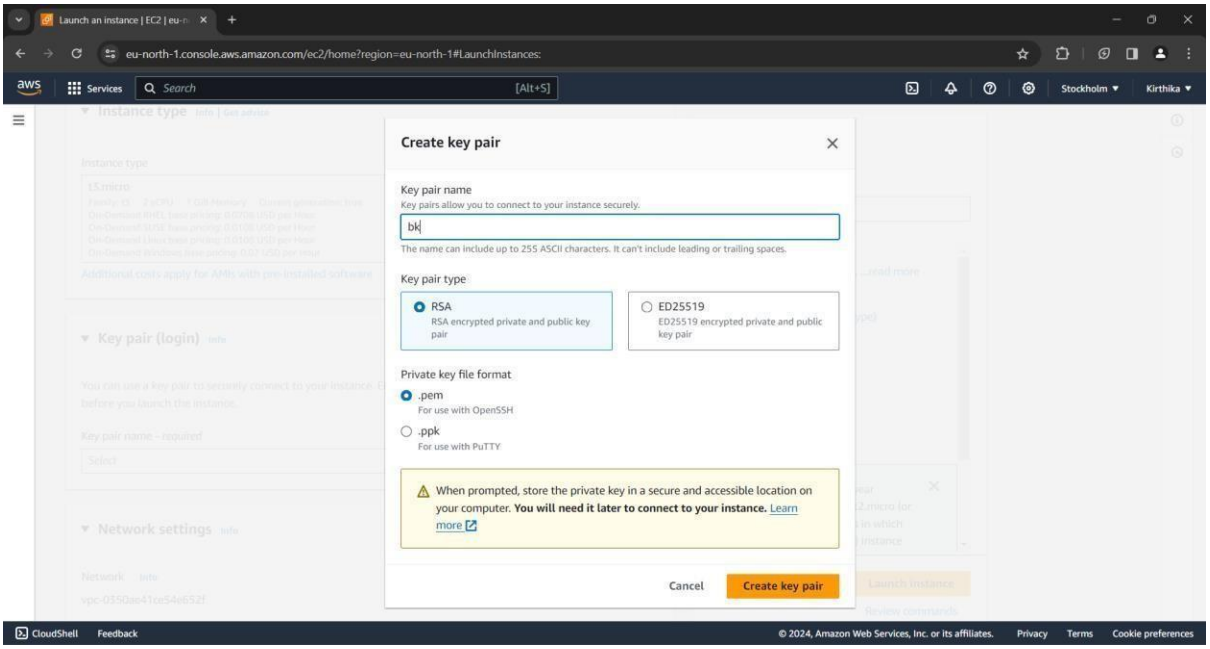
3. Provide a name for the instance for example **Jenkins Demo** and choose Ubuntu as the operating system.



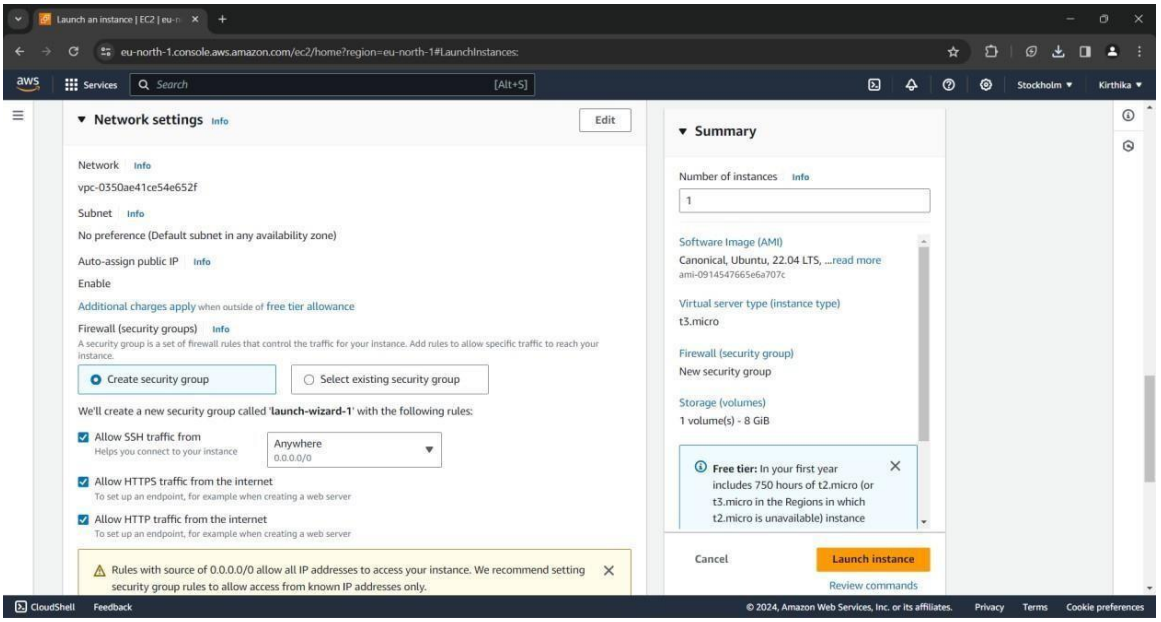
4. Create a new key pair if you haven't created one yet.



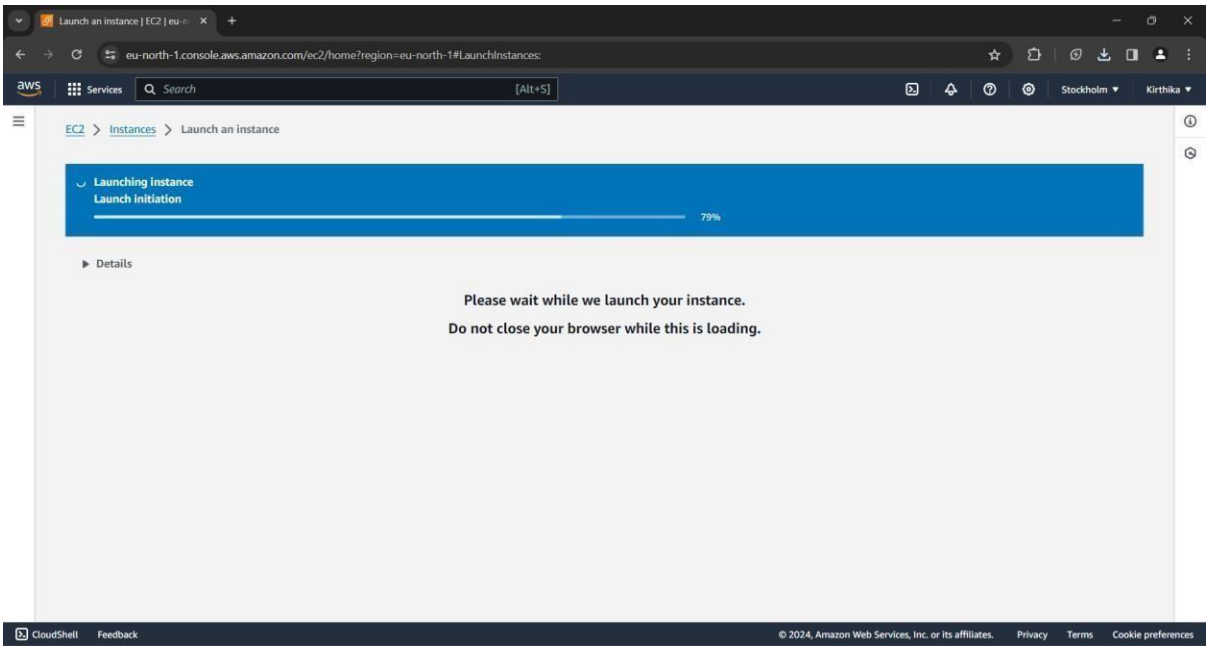
5. Give a suitable name for the key pair, select the key pair type as RSA and click on the **Create key pair** button at the bottom.



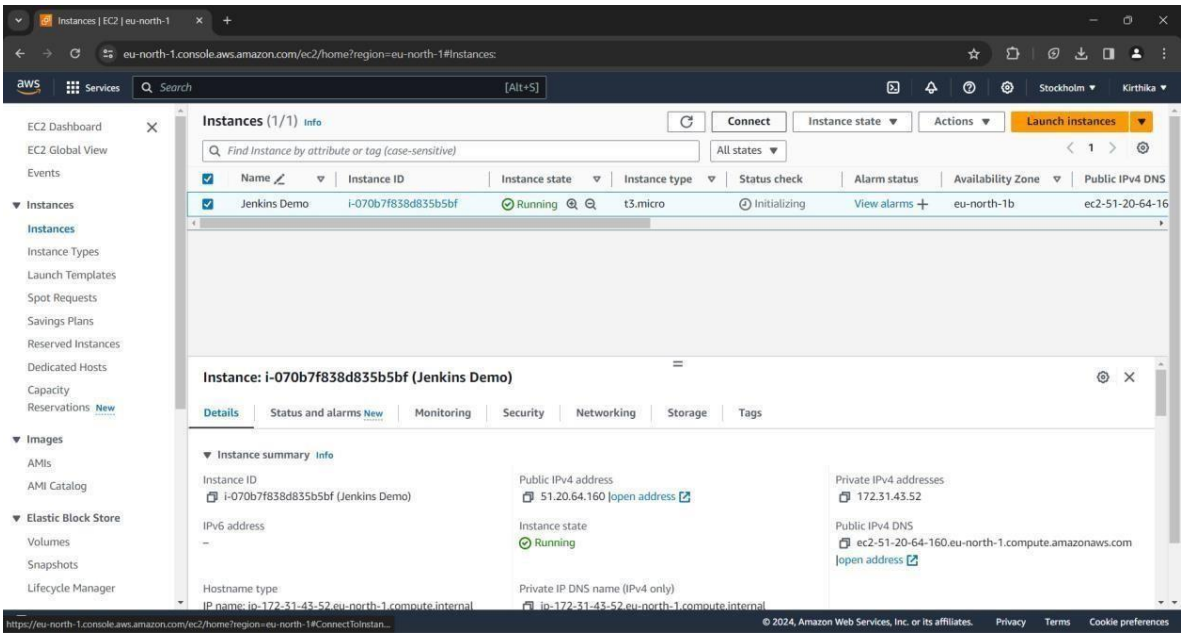
6. In the Network Settings tab select the following options.
    - Allow HTTPS traffic from the internet
    - Allow HTTP traffic from the internet



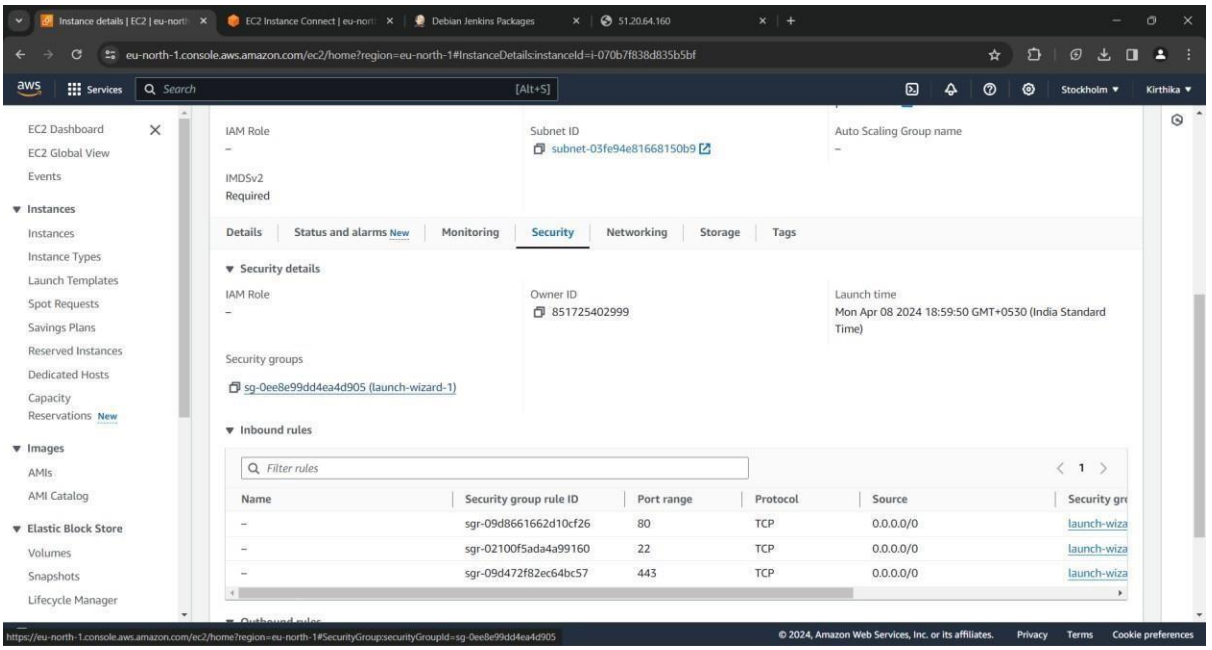
7. Wait while the instance is being launched. It may take some time.



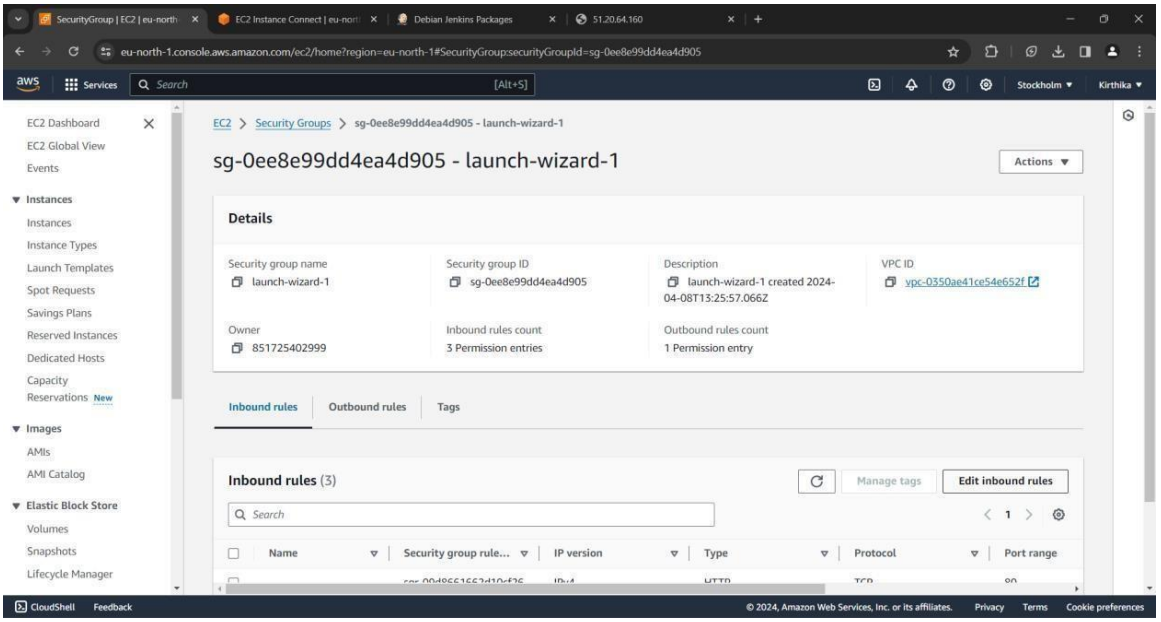
8. Select the Jenkins Demo instance and click on the **Connect** button.



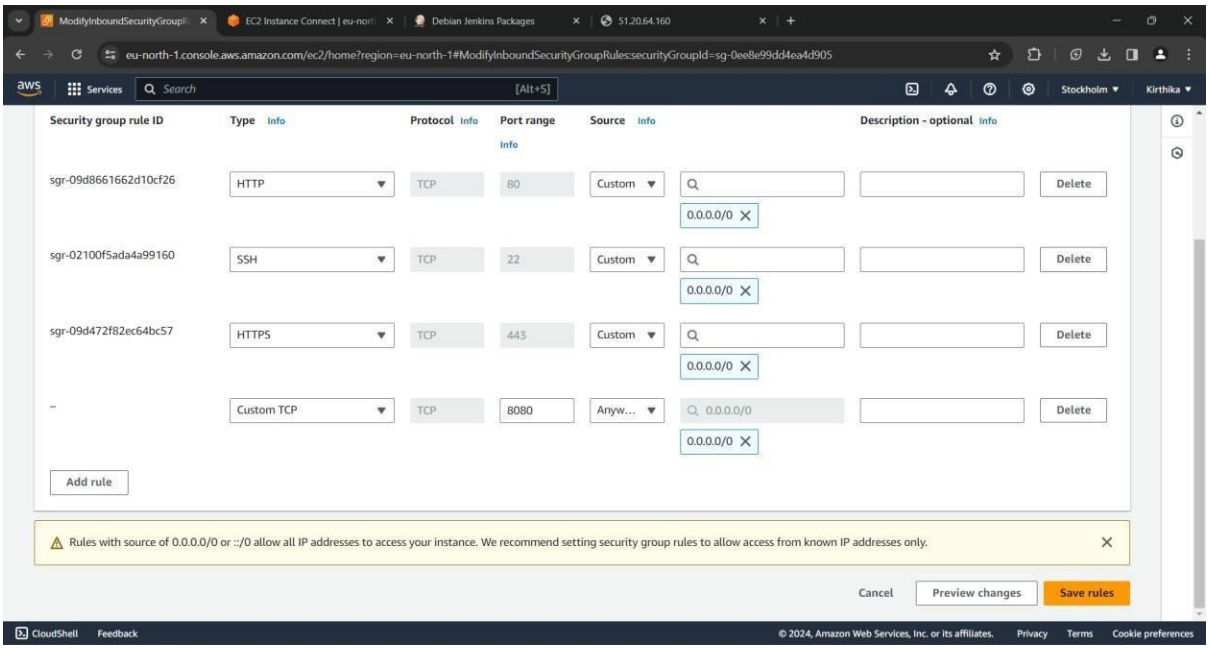
9. Click on the Security tab and choose the **Security Groups** option.



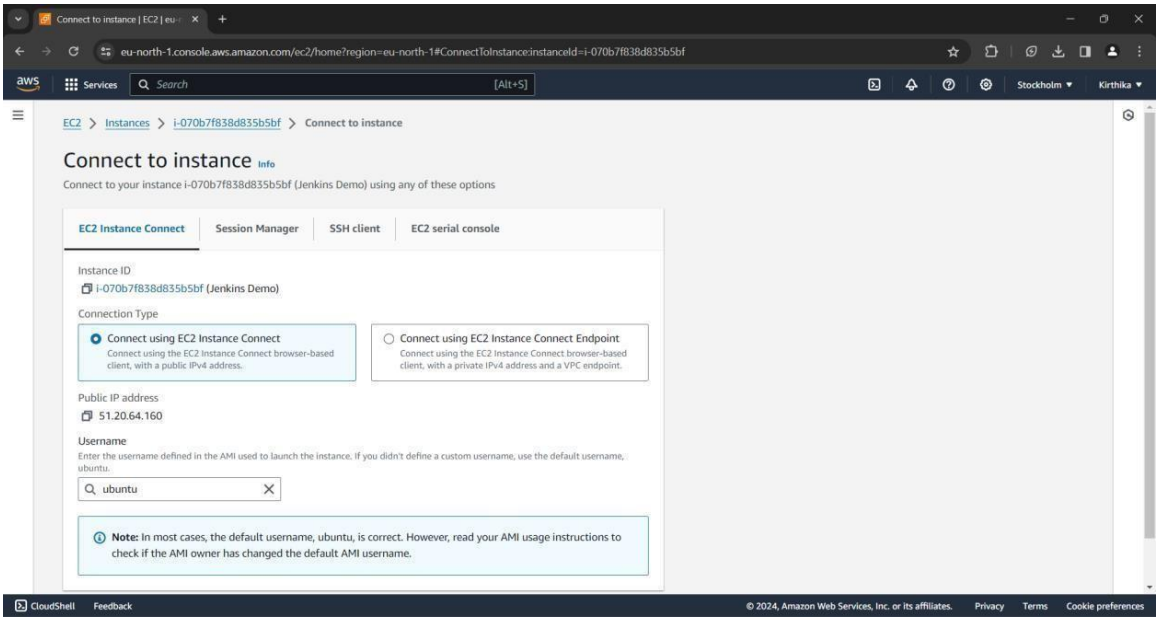
10. Click on the **Edit Inbound Rules** button.



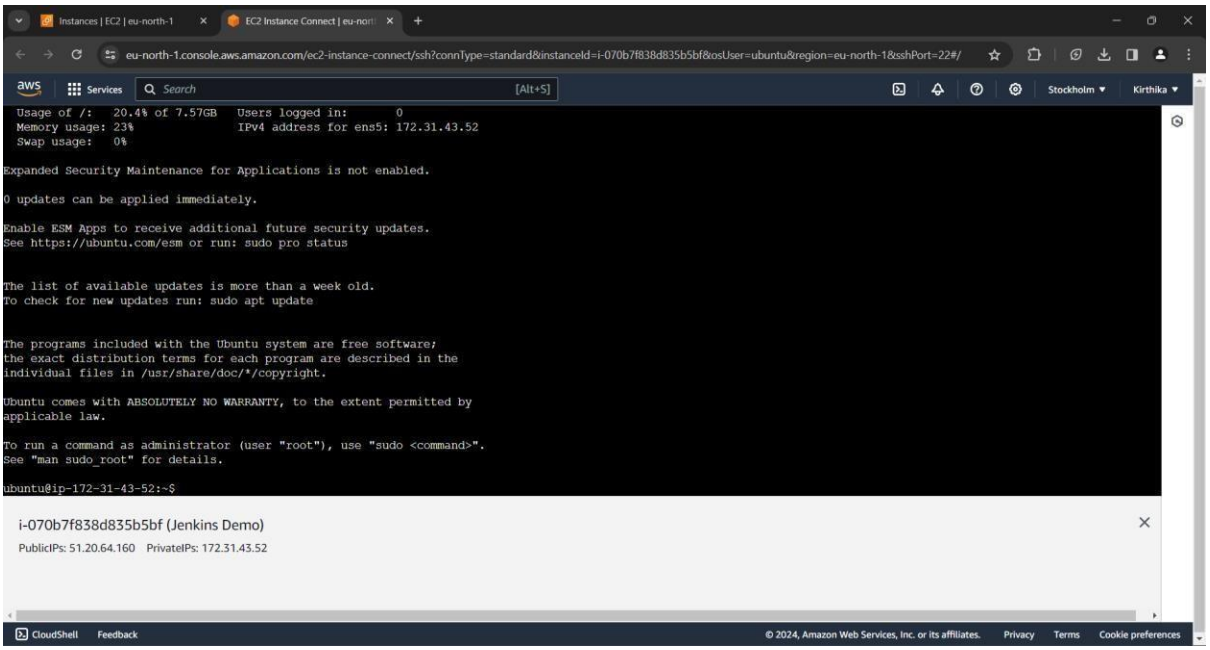
11. Click on **Add rule** at the left bottom, enter the port range as 8080 and choose 0.0.0.0/0



12. Click on the Connect button at the bottom after choosing Connect using **EC2 Instance Connect.**

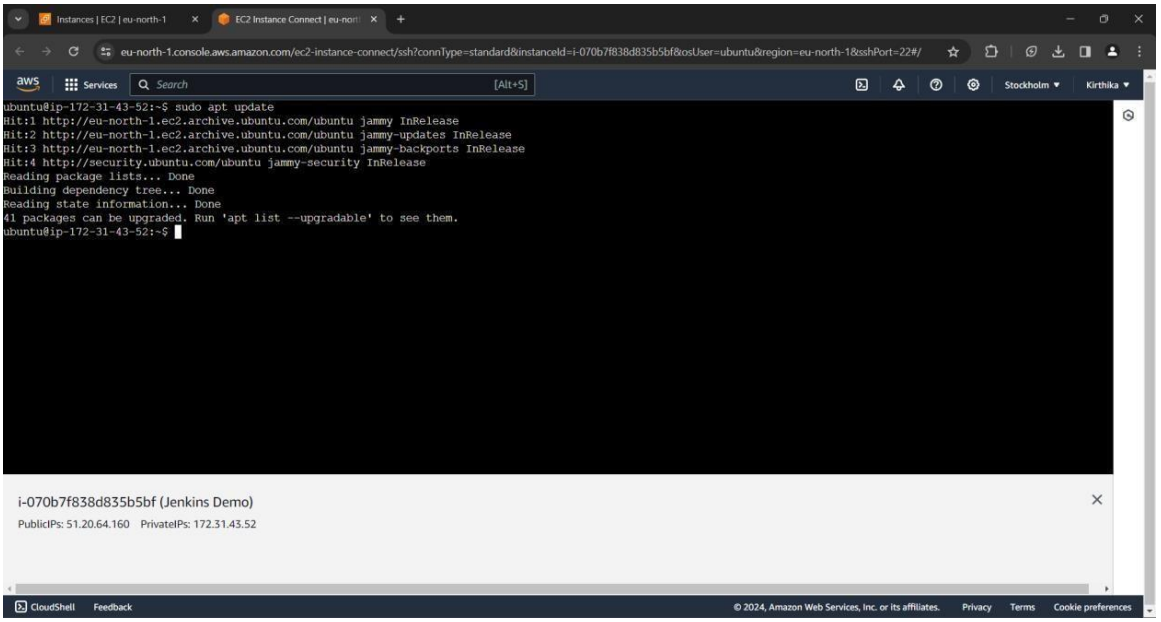


13. A terminal is opened on a new tab in the same browser.



STEP 3: Installing Jenkins on the instance.

1. Execute the following commands on the terminal to update the OS.
   $ clear
   $ sudo apt update



2. Execute the following command to install Jenkins.

   $ sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
   https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

   $ echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
   https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
   /etc/apt/sources.list.d/jenkins.list > /dev/null

   $ sudo apt-get update

   $ sudo apt-get install fontconfig openjdk-17-jre

   $ sudo apt-get install Jenkins

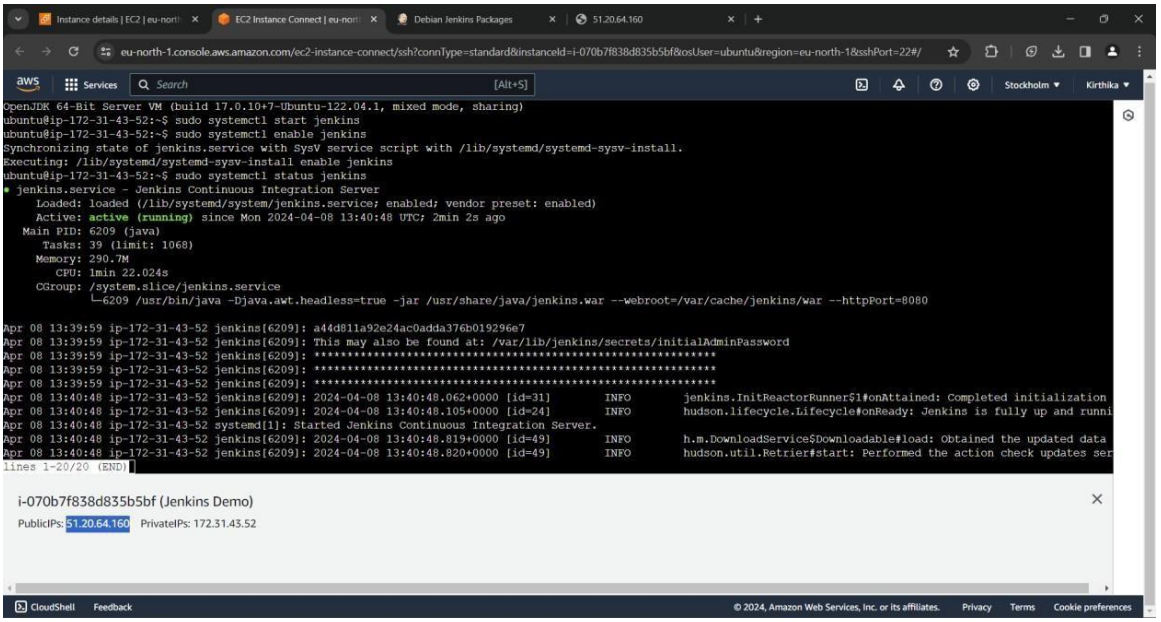
   These commands can also be found at https://pkg.jenkins.io/debian-stable/ for Debian releases.

3. The following commands will start Jenkins.

   $ sudo systemctl start Jenkins
   $ sudo systemctl enable Jenkins
   $ sudo systemctl status Jenkins

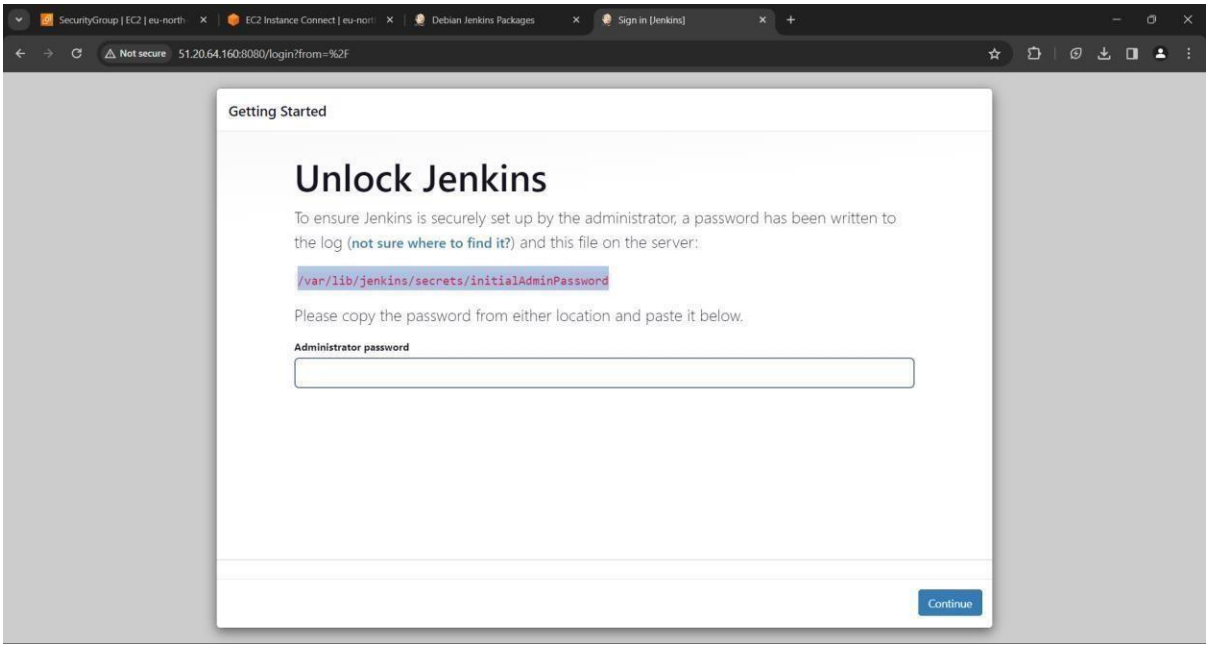
4. Copy the public IP address found below the terminal and copy the address. Open a new tab, paste the IP address and add **:8080** at the last.

   Example: 51.20.64.160:8080

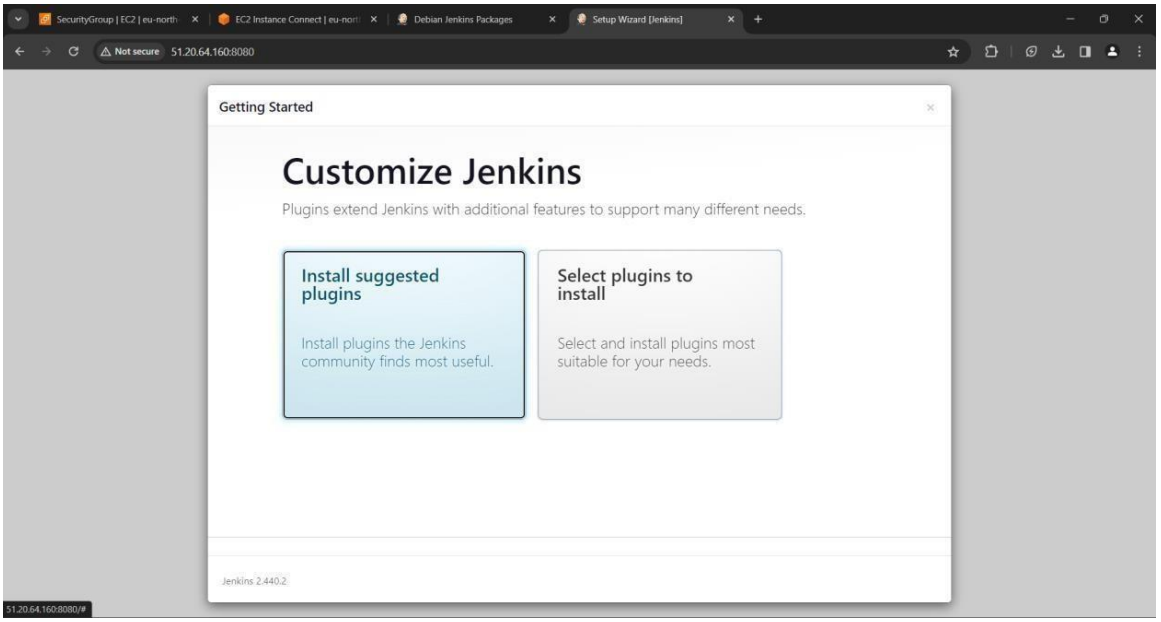


STEP 4: Configuring Jenkins

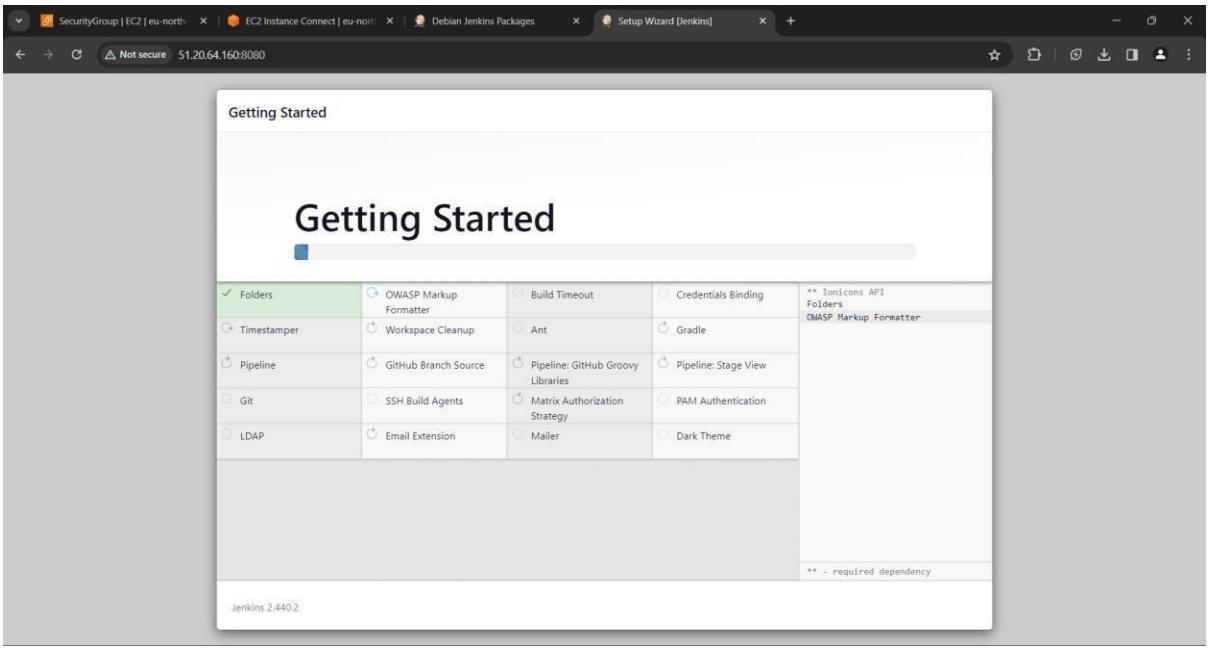
1. Jenkins will be initially locked. The password will be located at the location provided on the screen. Copy it.
2. Go back to the terminal and execute
   $ sudo cat /var/lib/Jenkins/secrets/initialAdminPassword
3. Copy the password from the terminal, paste it on the Jenkins webpage and press Continue.



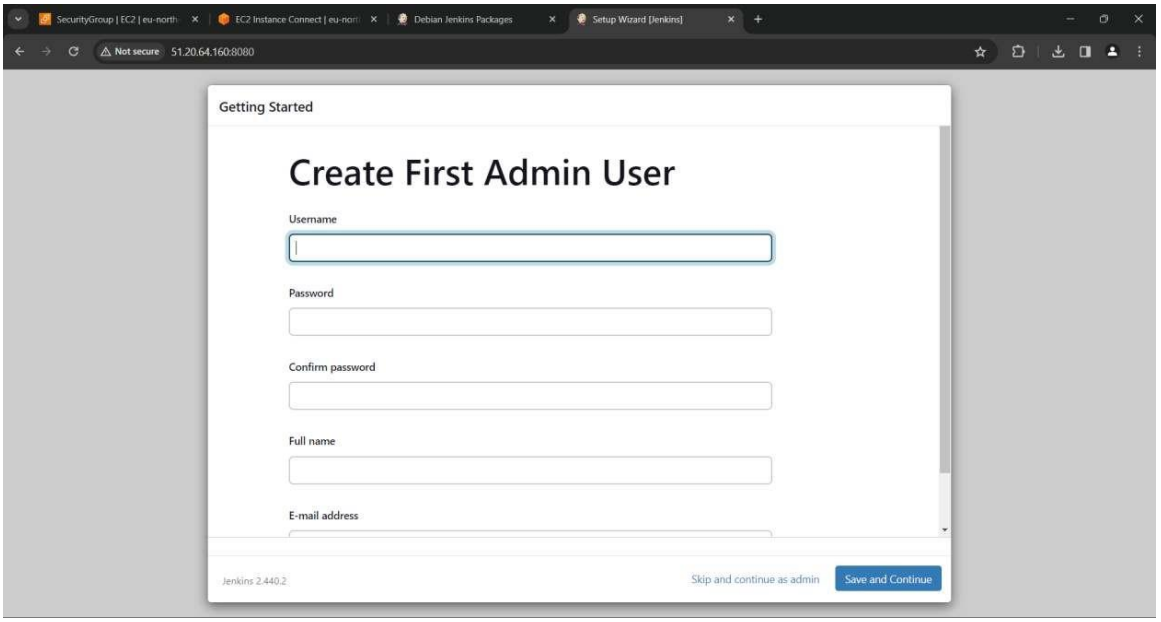
4. Select Installed suggested plugins to install the most used plugins in Jenkins.



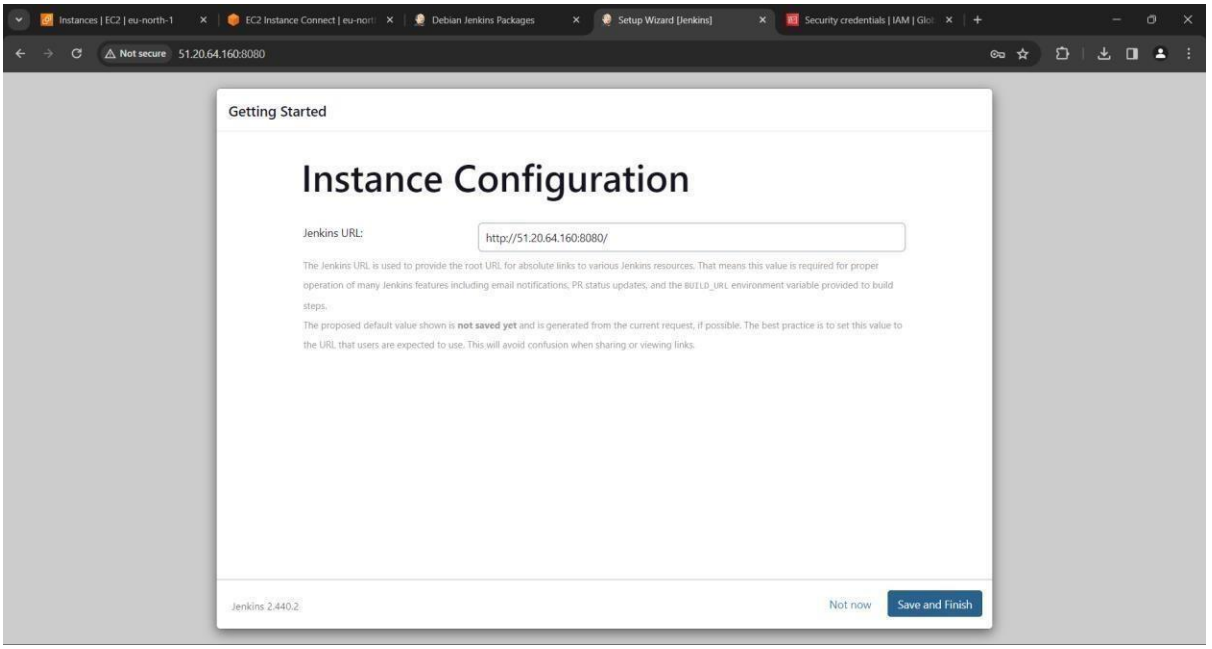
5. The installation will take some time to complete.



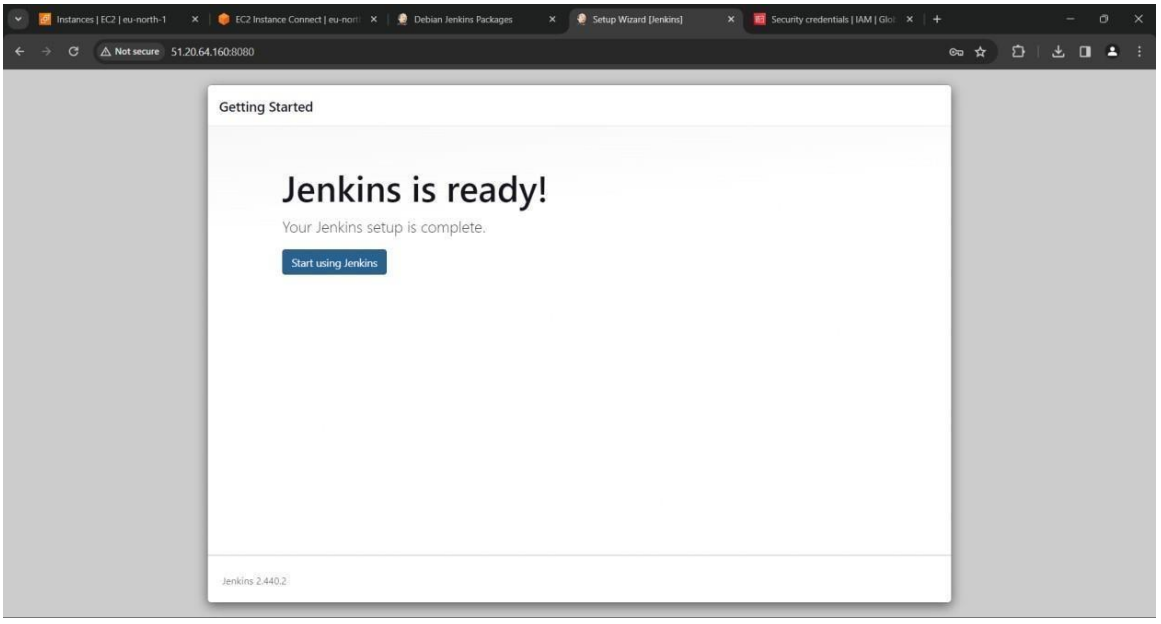
6. Create a user login on the next page. Provide the necessary information and press **Save and Continue.**



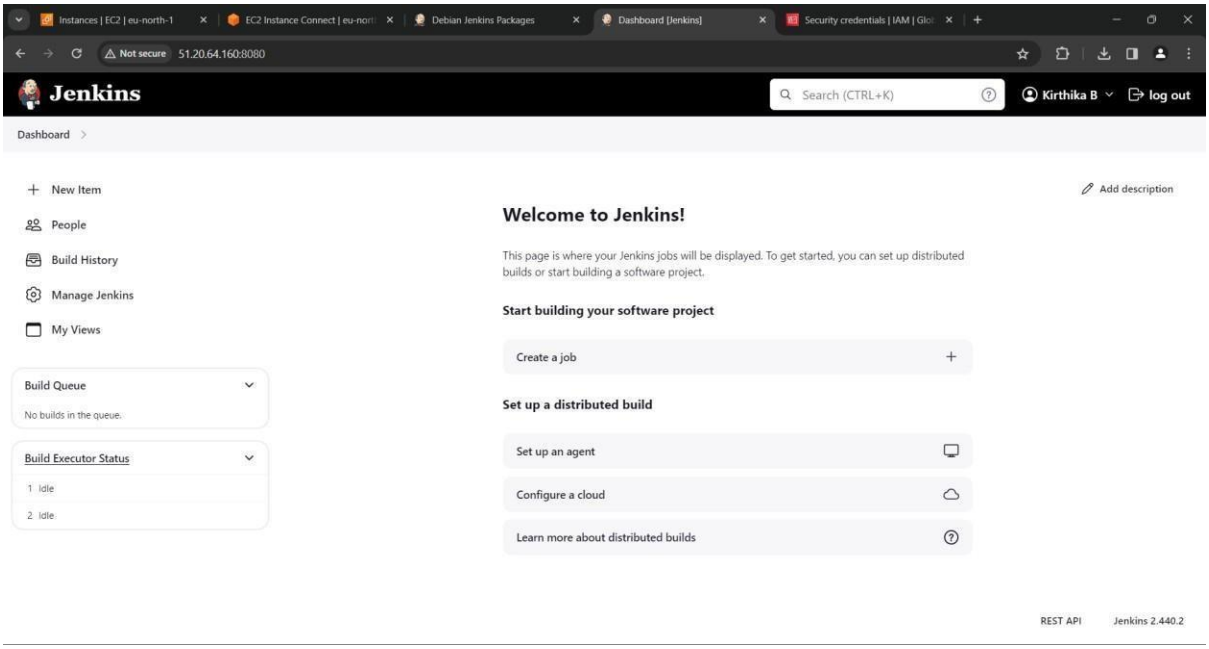
7. Check whether the Jenkins URL is the same as the one mentioned in the address bar and press **Save and Finish.**



8. Now Jenkins has been successfully configured on the cloud. Click on Start using Jenkins to work with it.



9. The Jenkins web page would look like below. Jobs can be built here.



RESULT:

Thus, Jenkins has been installed on the cloud.

Ex. No: 4 **CREATE CI PIPELINE USING JENKINS**

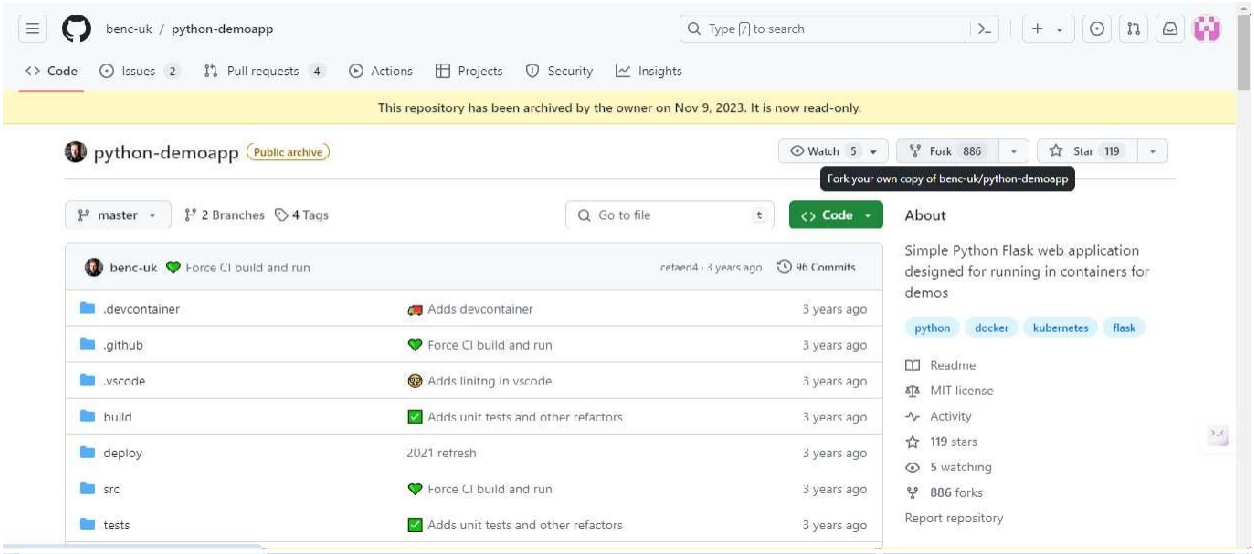
Date:

AIM:

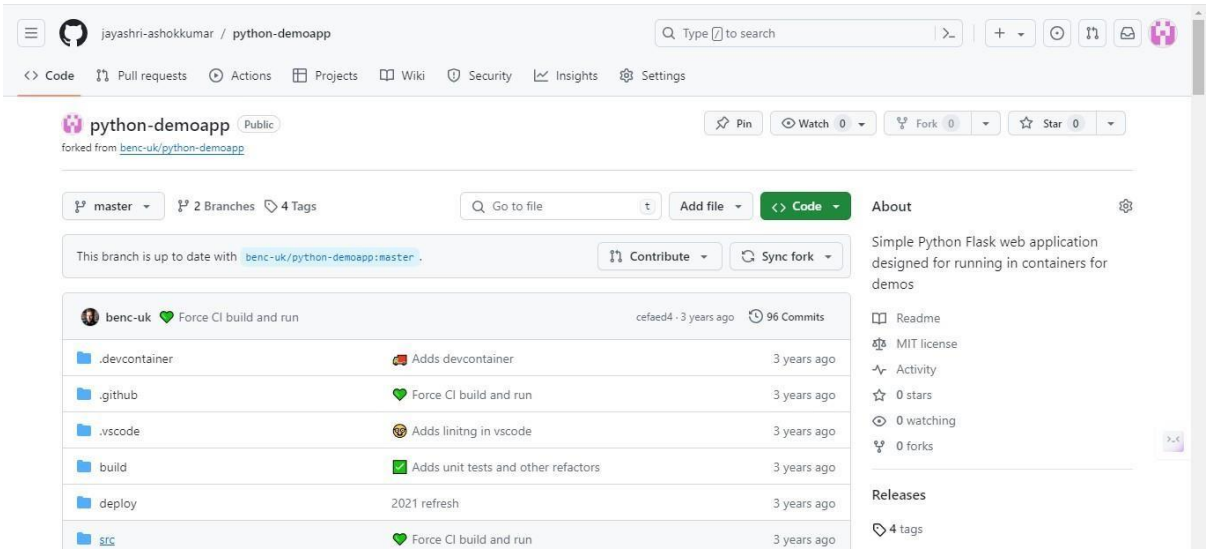
To create a CI pipeline using Jenkins.

PROCEDURE:

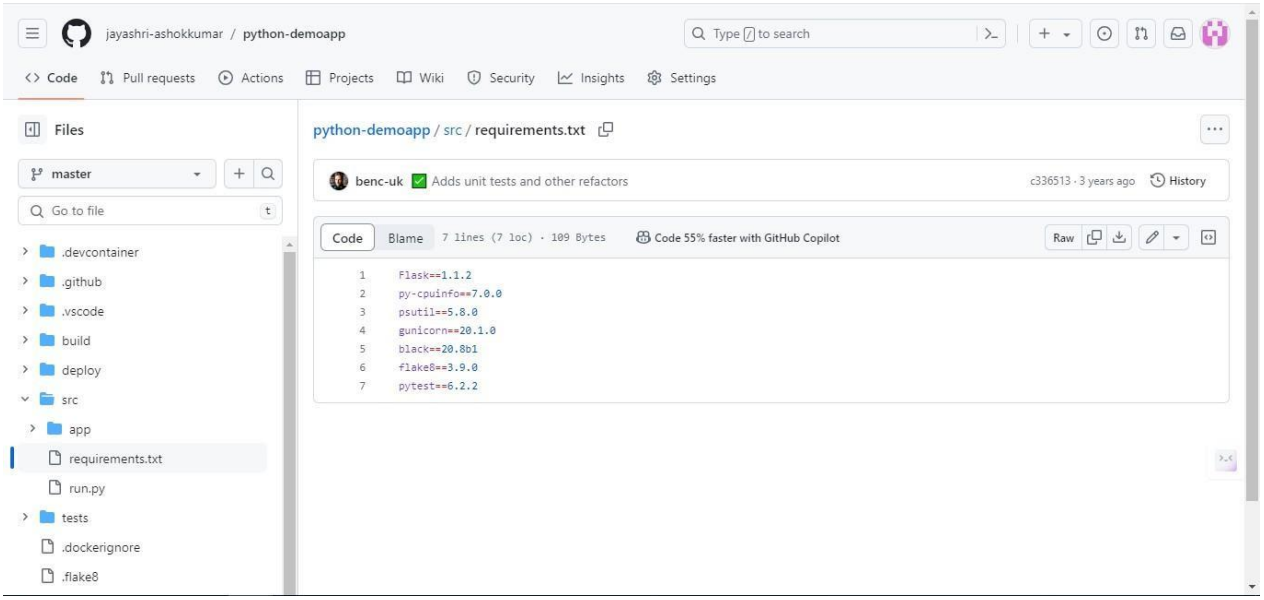
STEP 1: Click and open this GitHub repository (https://github.com/benc-uk/python-demoapp.git). And also click fork to make the changes in your own account.
(Note: The username and password in GitHub, Jenkins and Docker Hub website should be the same.)



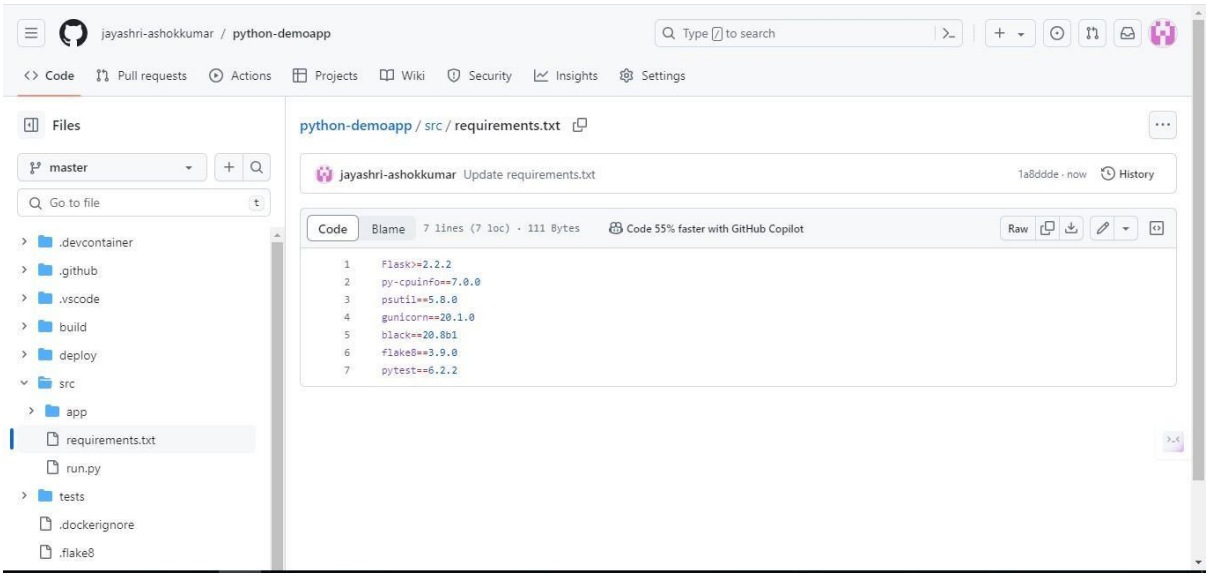
STEP 2: Now you have moved into your own GitHub account.



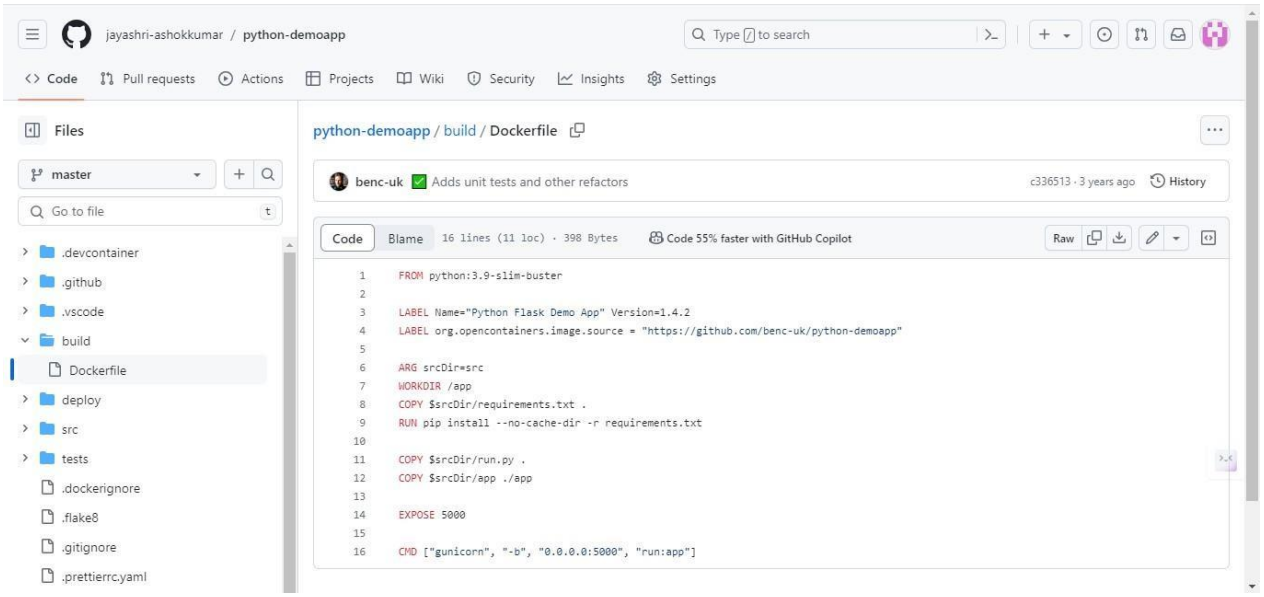
STEP 3: Click the src directory, choose requirements.txt and you will be able to see the below code.



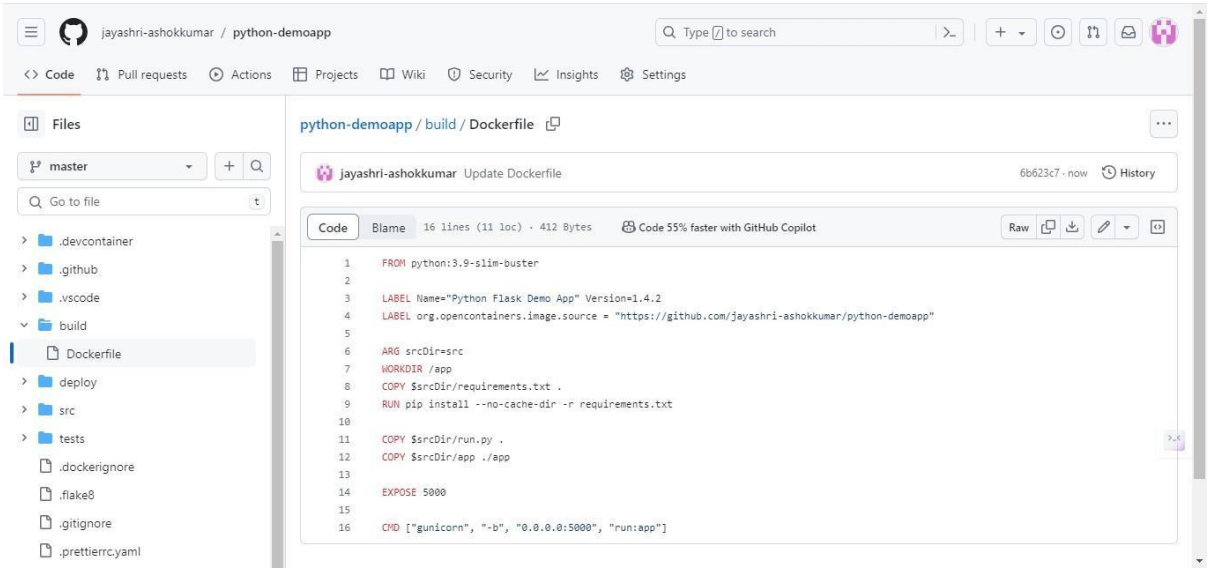
STEP 4: Now, you can make the below changes in the code.



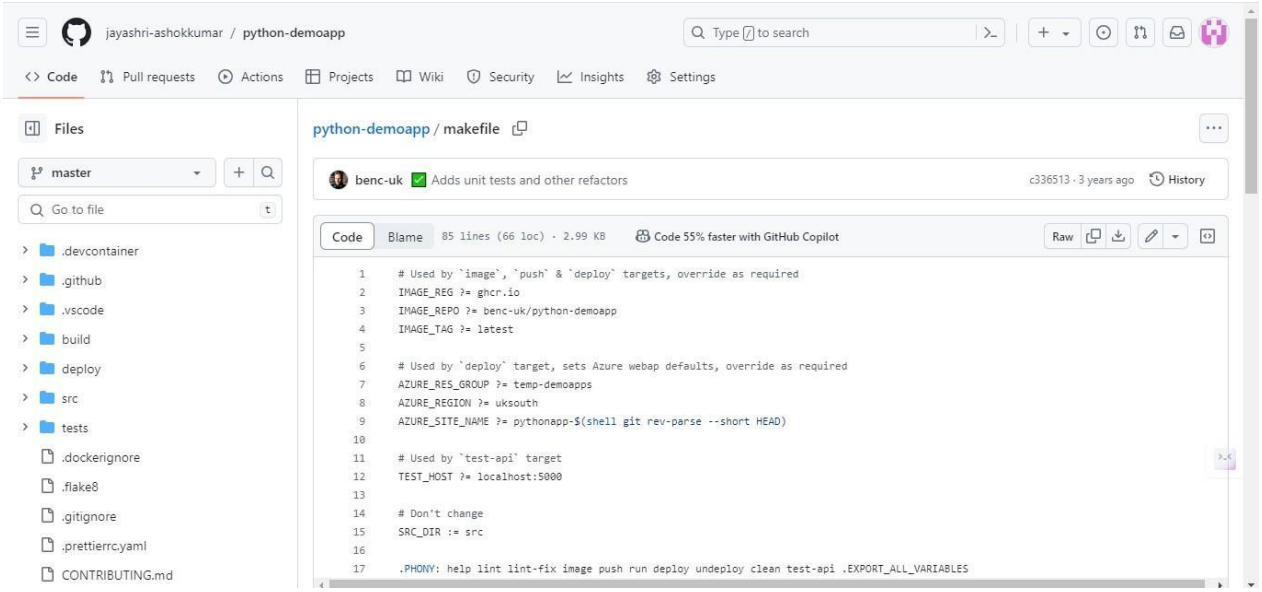
STEP 5: Click build directory and choose Dockerfile and you will be able to see the below code.



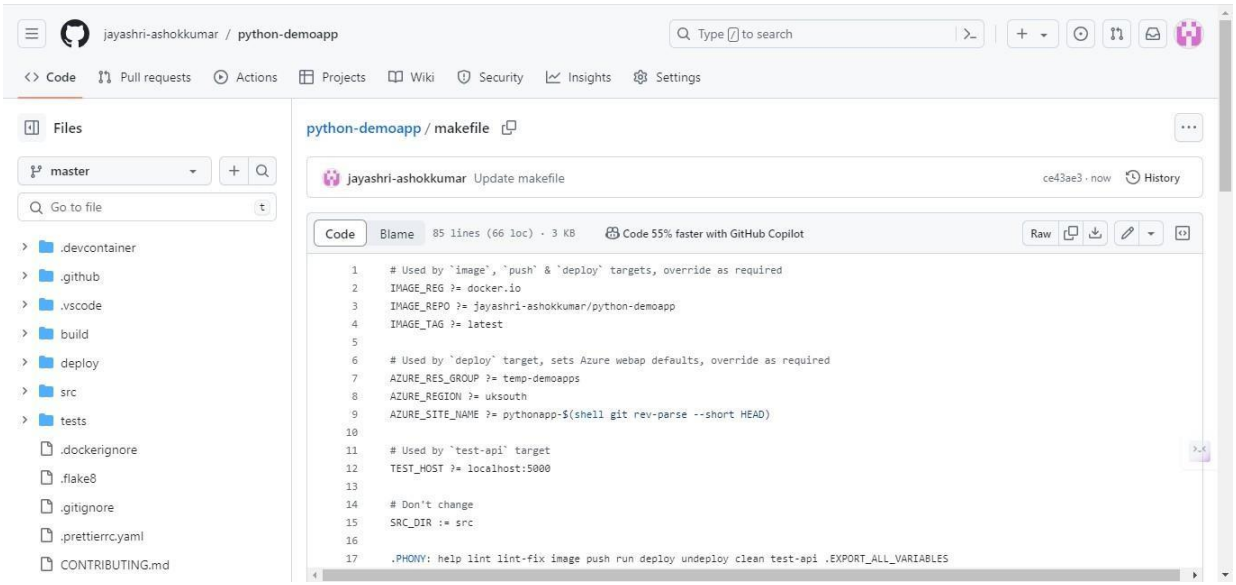
STEP 6: Replace the above link with your own GitHub repository link.



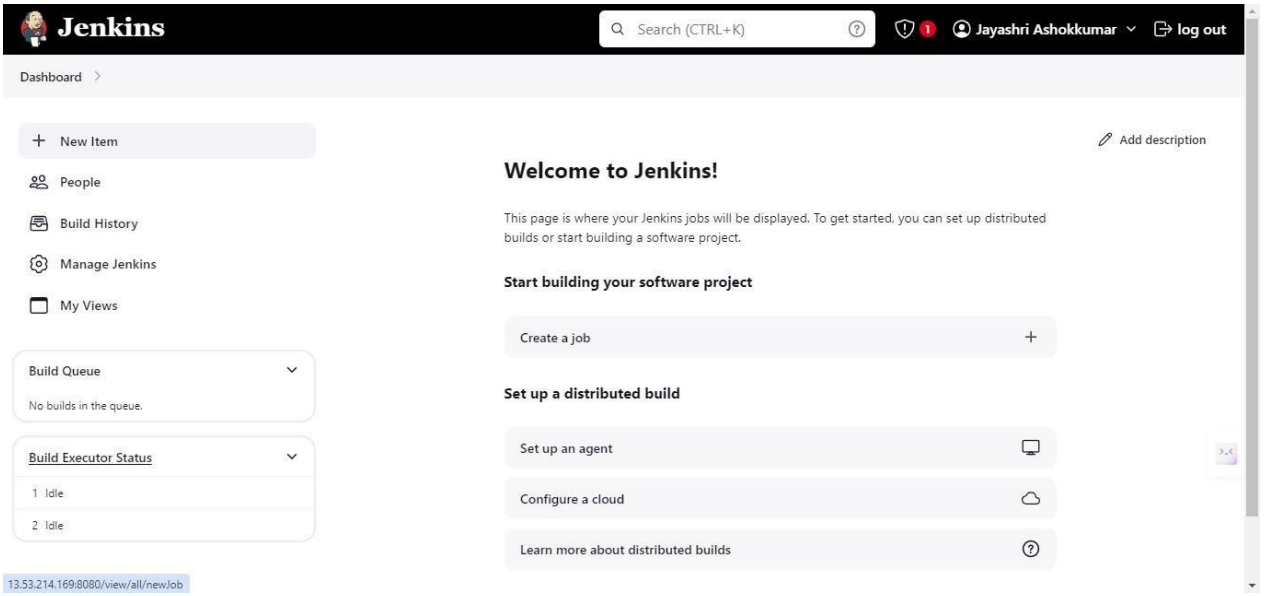
STEP 7: Click on makefile, and you will be able to see the below code.



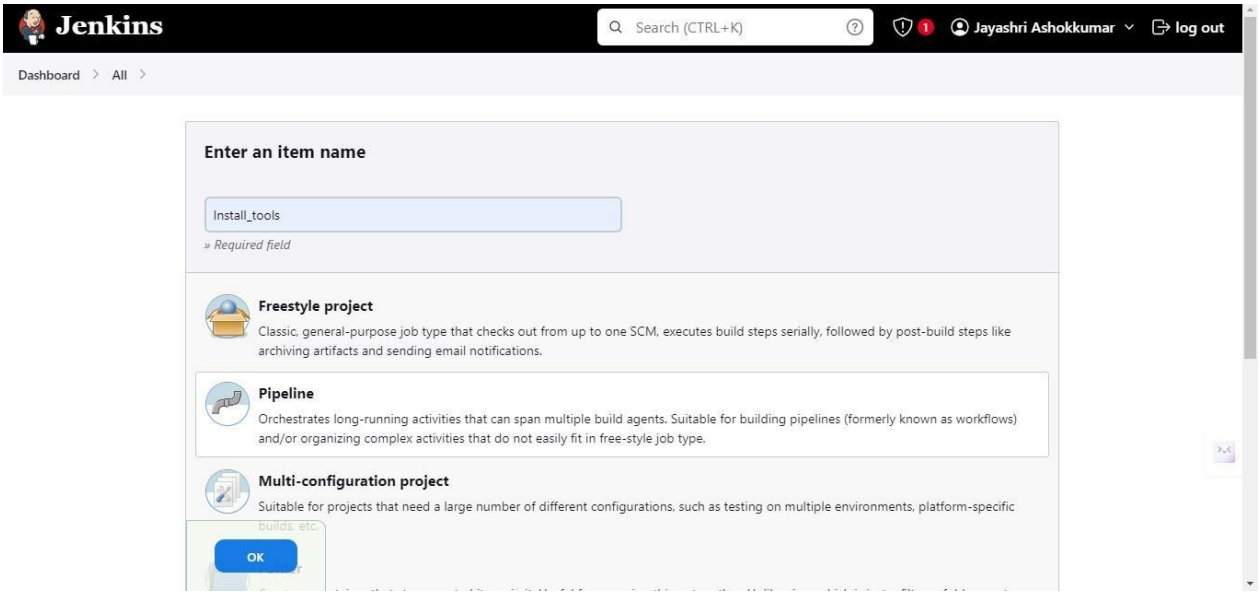
STEP 8: Replace the above name in the code with your own GitHub name and also replace the ghcr.io with docker.io.



STEP 9: Open the Jenkins page, and click on the new item that is present in the dashboard.



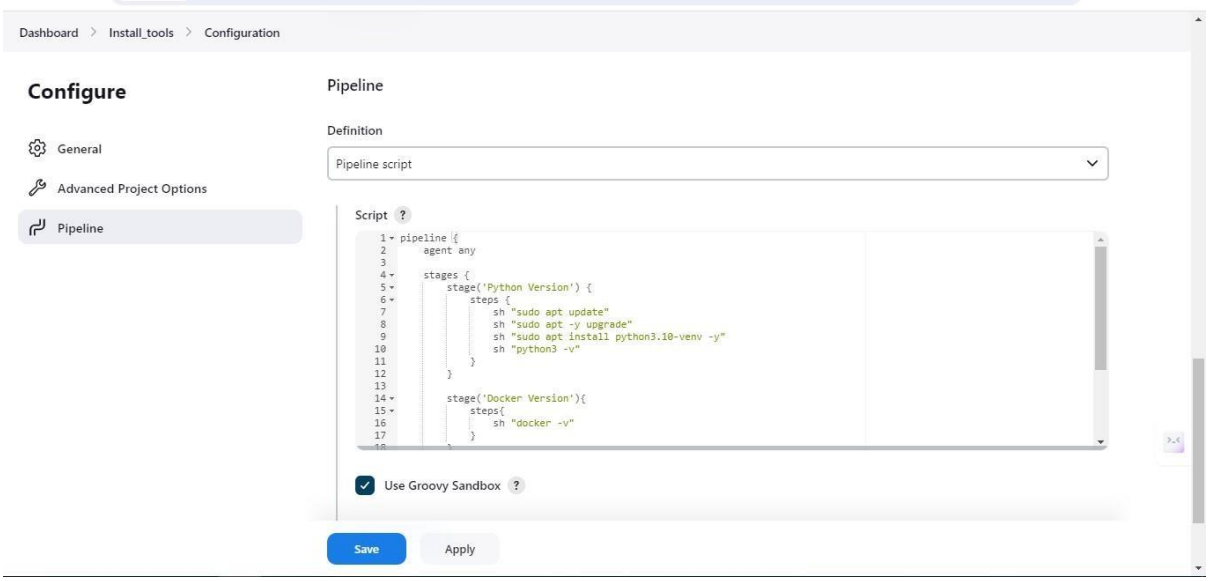
STEP 10: Now Enter the item name and select pipeline, then click on OK.



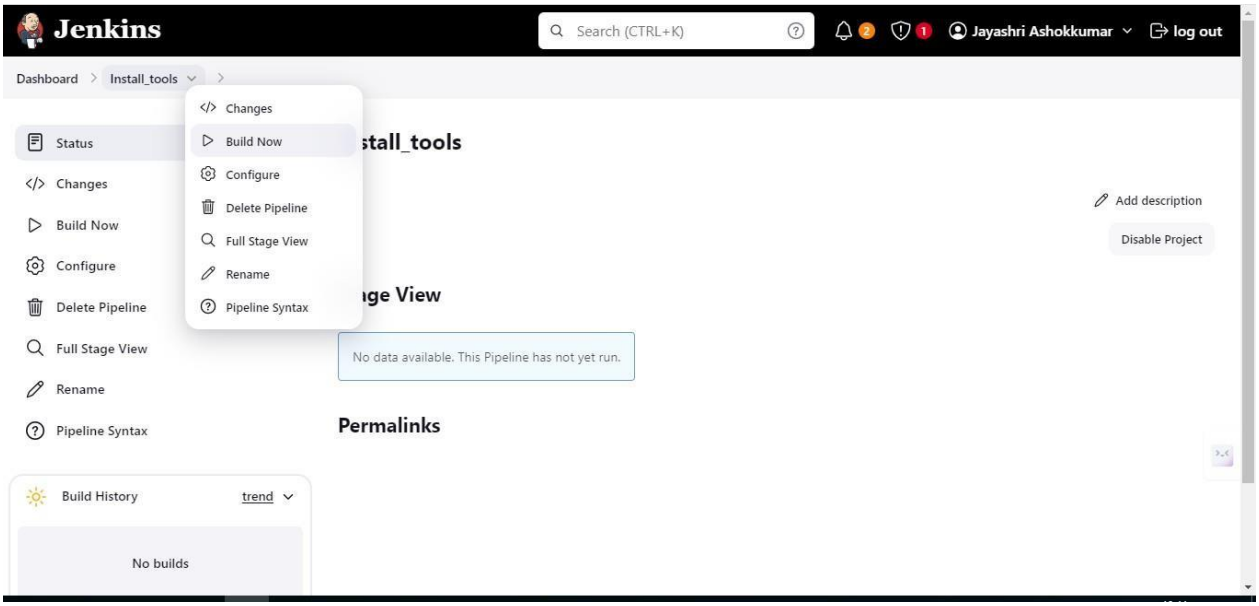
STEP 11: Click > Install_tools > Configure > Pipeline. In the definition section of Pipeline, Choose Pipeline script and the script area will be visible in which you will type the below code.

```
pipeline {
    agent any
          stages {
        stage('Python Version') {
            steps {
                sh "sudo apt update"
                sh "sudo apt -y upgrade"
                sh "sudo apt install python3.10-venv -y"
                sh "python3 -v"
            }
}
        stage('Docker Version'){
            steps{
                sh "docker -v"
            }
        }
        stage('Install Make'){
            steps{
                sh "sudo apt install make -y"
                sh "sudo apt install make-guile -y"
            }
        }
    }
}
```

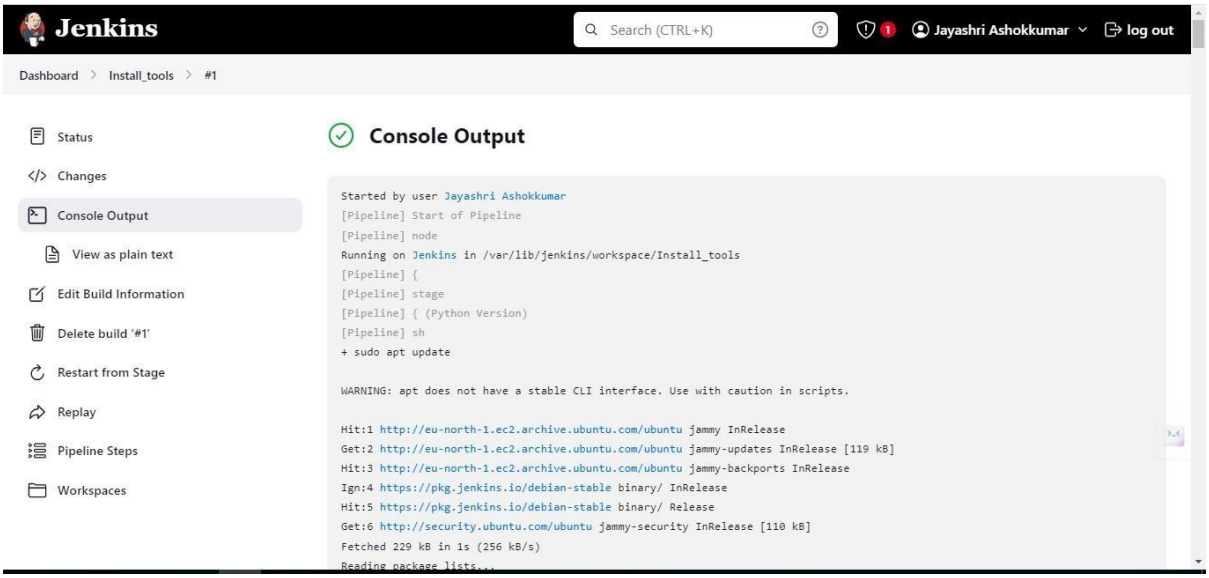
Now, Click on Save.

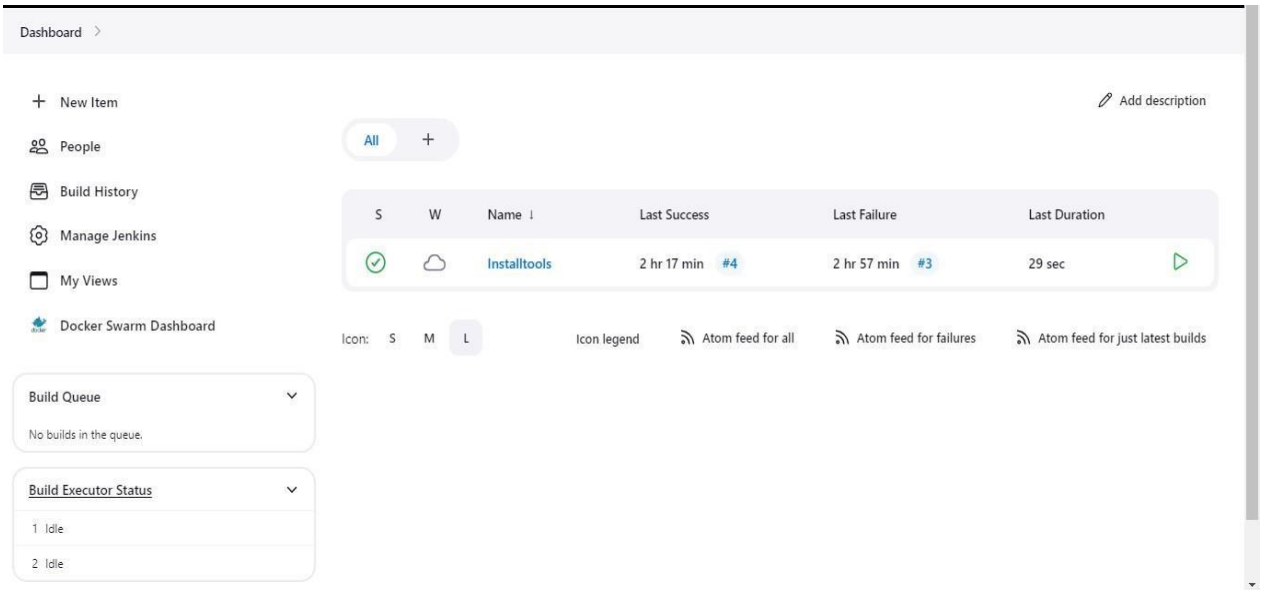


STEP 12: Click on the Install_tools and select the build now.

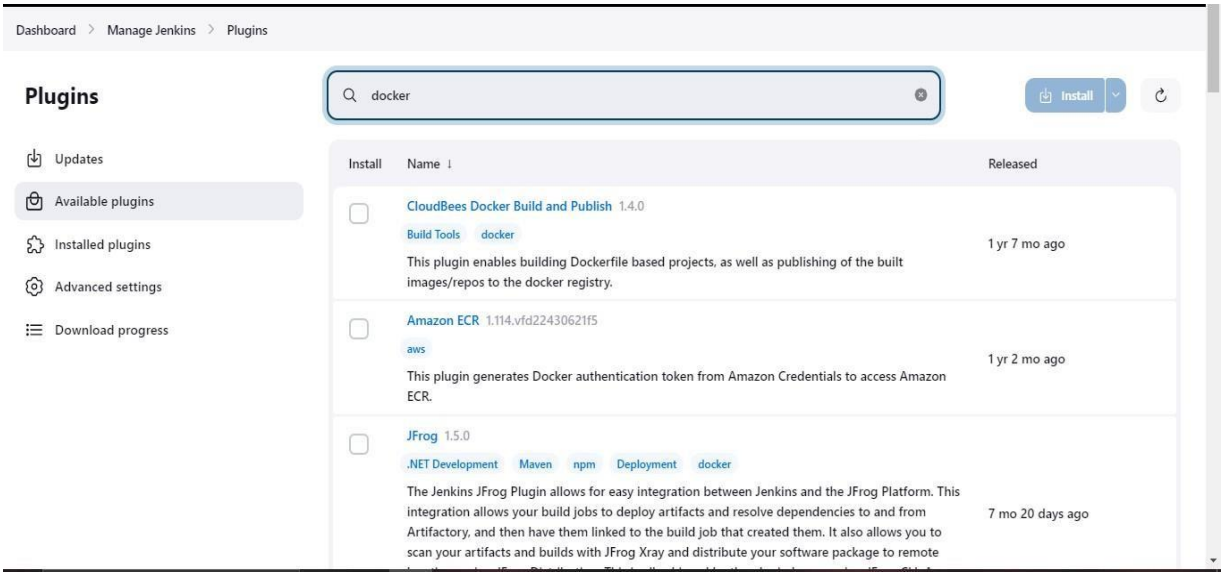


STEP 13: In the Build history section, click on the builds then you will able to see the console output as 'FINISHED: SUCCESS'. Now The Pipeline is built.

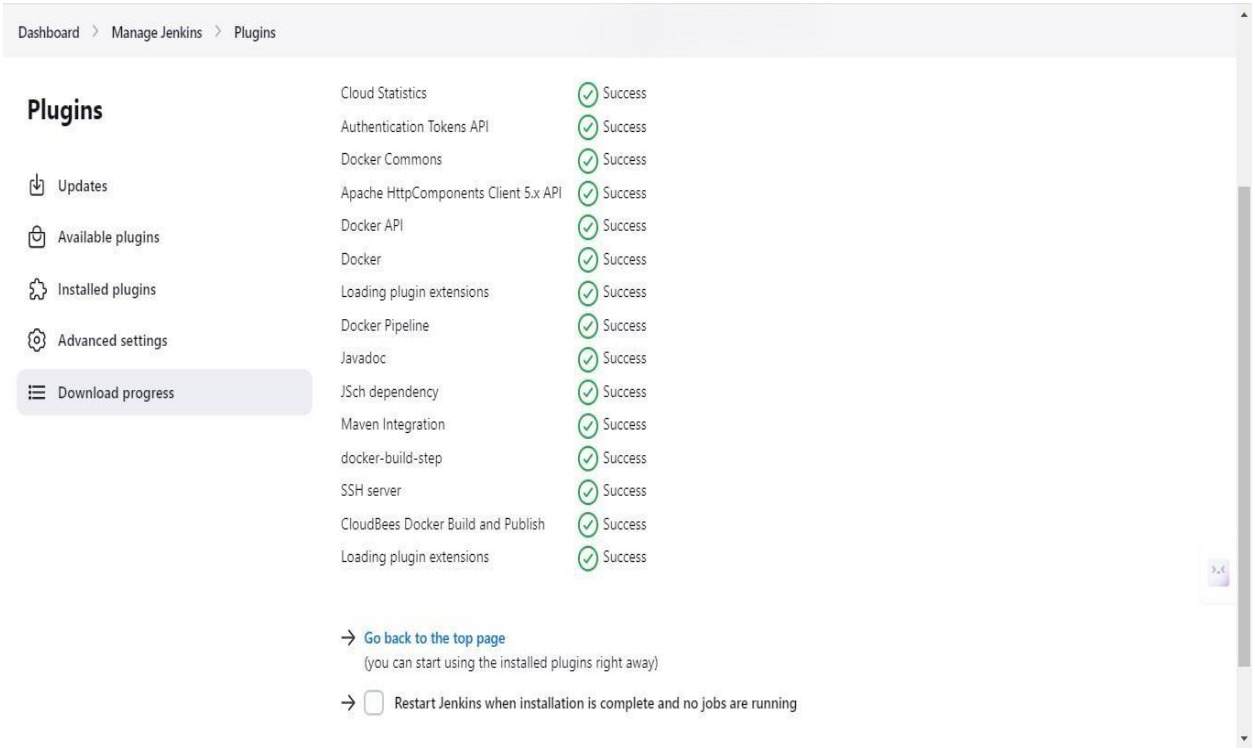




STEP 14: Click > Dashboard > Manage Jenkins> Plugins. In Plugins, Select Available Plugins. Here, search for docker and select the associated plugins (Docker, docker pipeline, docker build step, CloudBees Docker Build and Publish) then install it.



STEP 15: In Download progress, you will be able to see the set of plugins that were installed.



STEP 16: Run the following commands in the AWS ubuntu terminal.

1. To Create the Docker Group:
   $sudo groupadd docker

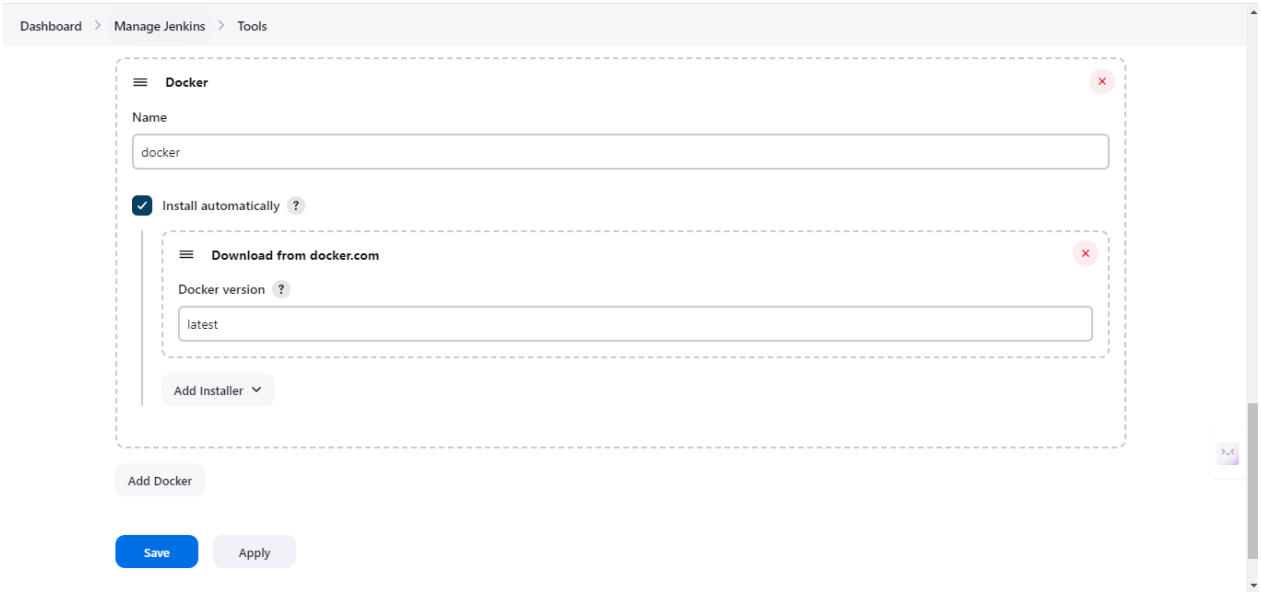
2. To Add the Jenkins User to the Docker Group:
   $sudo usermod -aG docker jenkins
   (Replace 'jenkins' with the actual username of the Jenkins user)

3. To Verify the Group Membership:
   $groups Jenkins

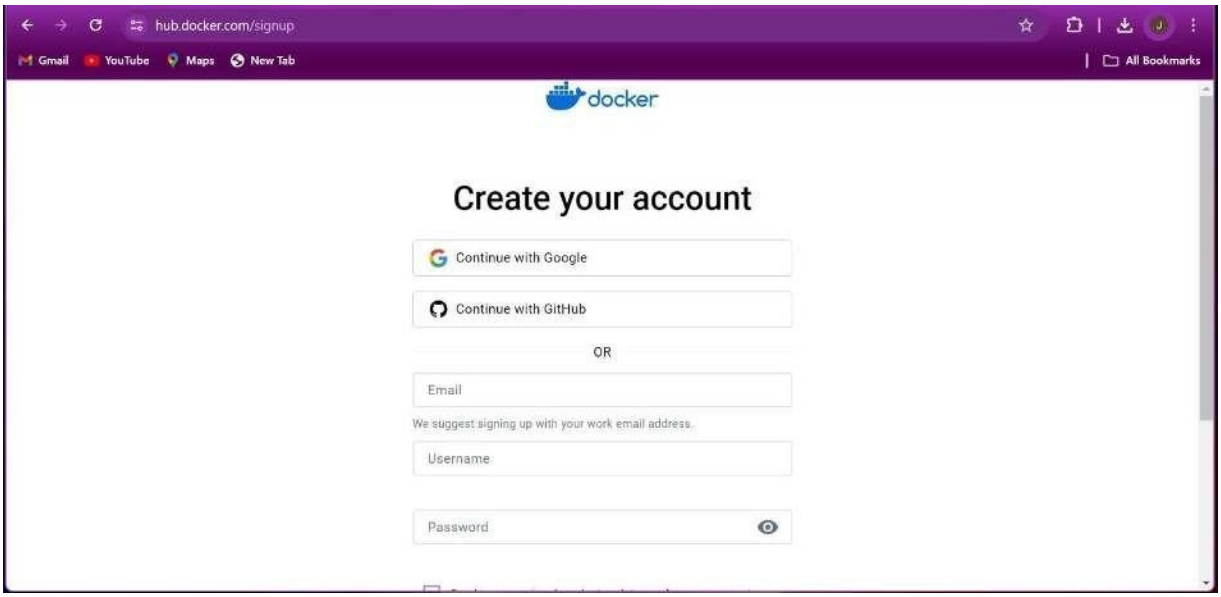
4. To install the docker:
   $sudo apt install docker.io

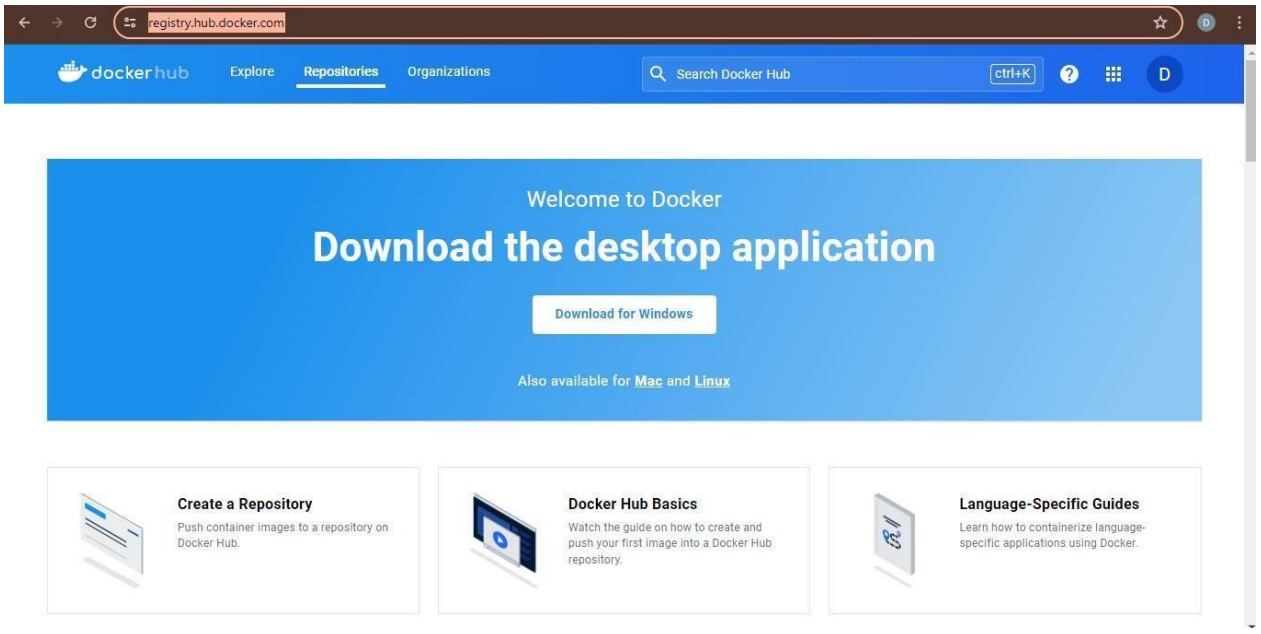
5. To restart jenkins:
   $sudo systemctl restart Jenkins

STEP 17: Click > Manage Jenkins> Tools. Scroll for the docker section and perform the following actions in the image.

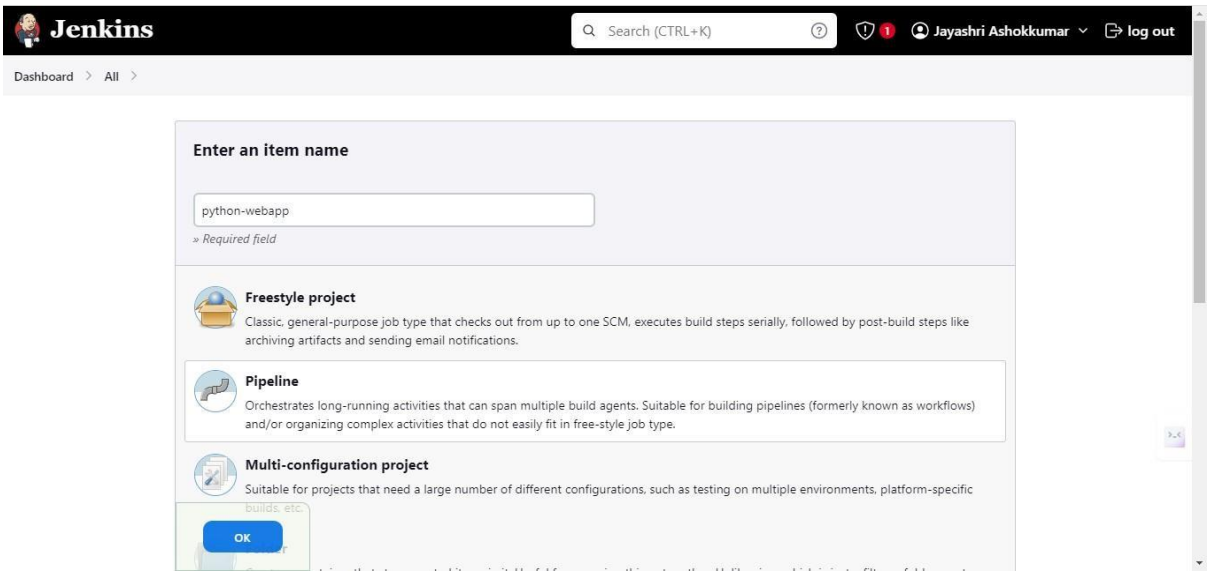


STEP 18: Open Browser and search for docker hub, then create an account in docker hub.

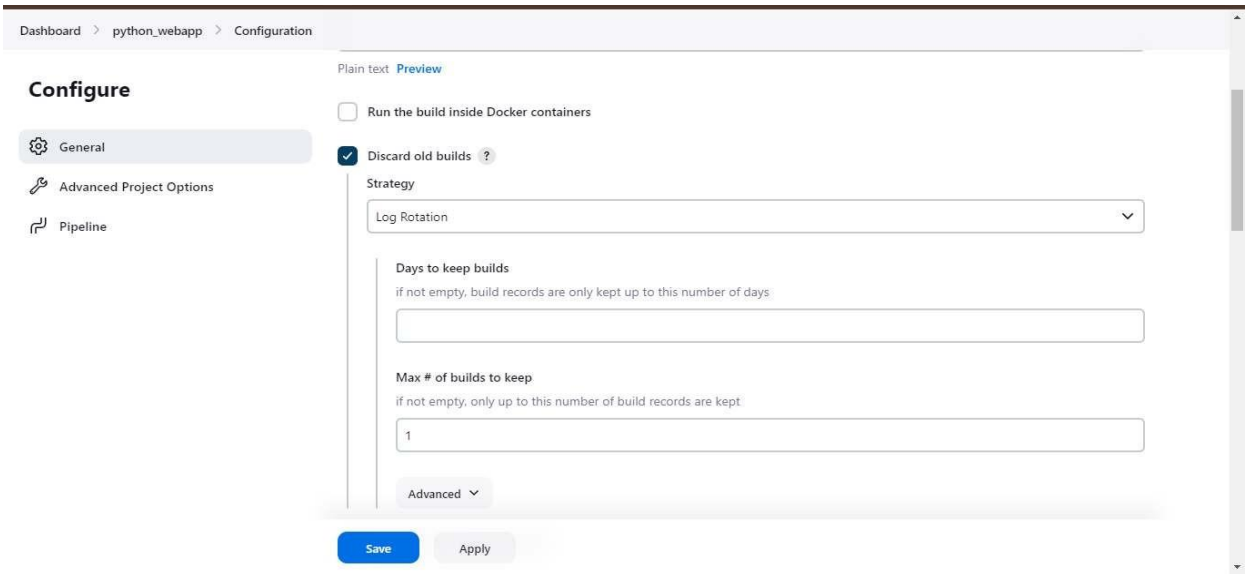




STEP 19: Click on the new item in the Jenkins dashboard. Now Enter the item name and select pipeline, then click on OK.

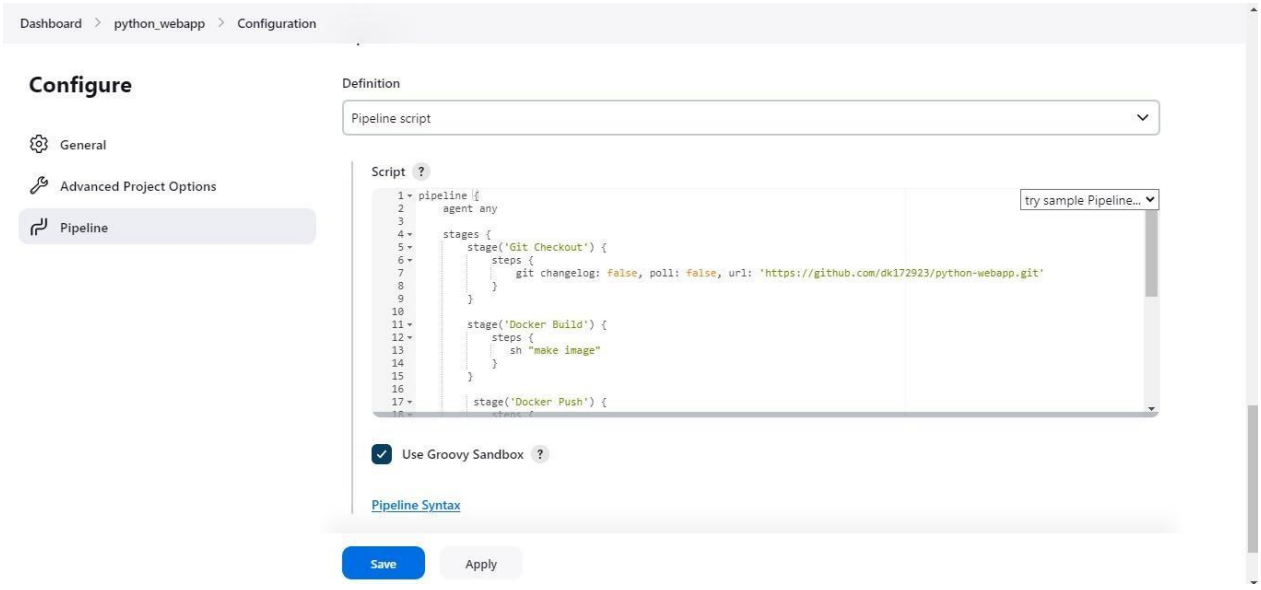


STEP 20: Click > dashboard > python-webapp > configure. In the general section, select 'Discard old builds' and set the max of builds to 1.



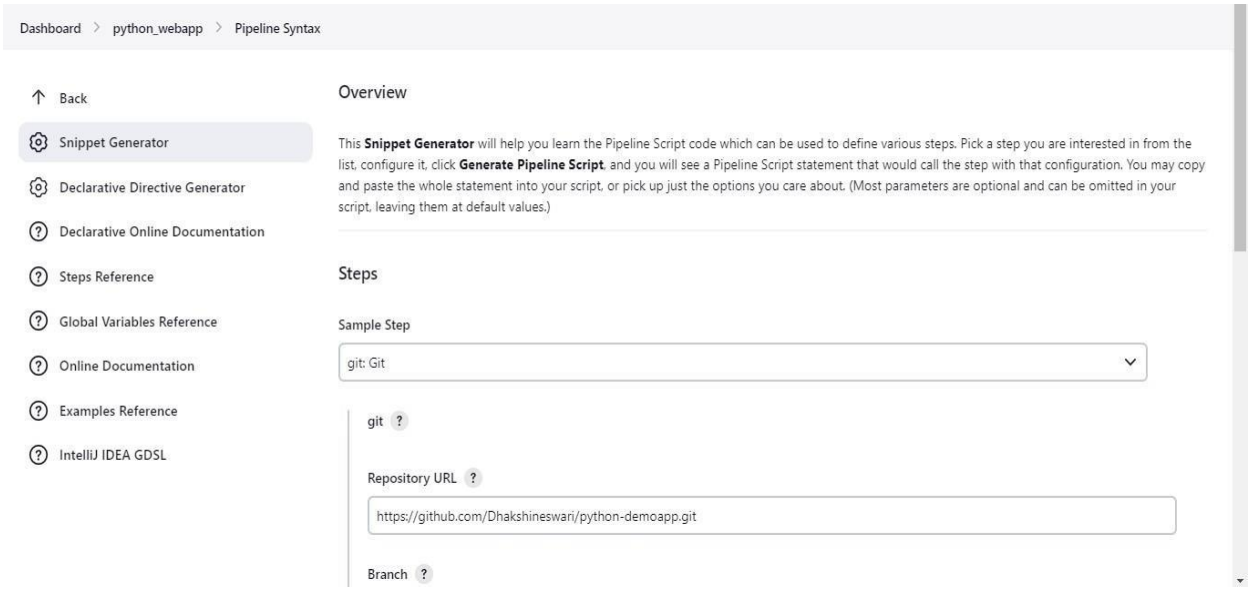
STEP 21: Scroll down for Pipeline. In the Script area, run the below code.

```
pipeline {
   agent any
     stages {
       stage('Git Checkout') {
          steps {
             git changelog: false, poll: false, url:''            }
       }
       stage('Docker Build') {
          steps {
            sh "make image"
          }
       }
        stage('Docker Push') {
          steps {
            script{
              withDockerRegistry(credentialsId:' ', toolName:' '  ) {
                   sh "make push"
                }
            }
          }
        }
       }
}
```

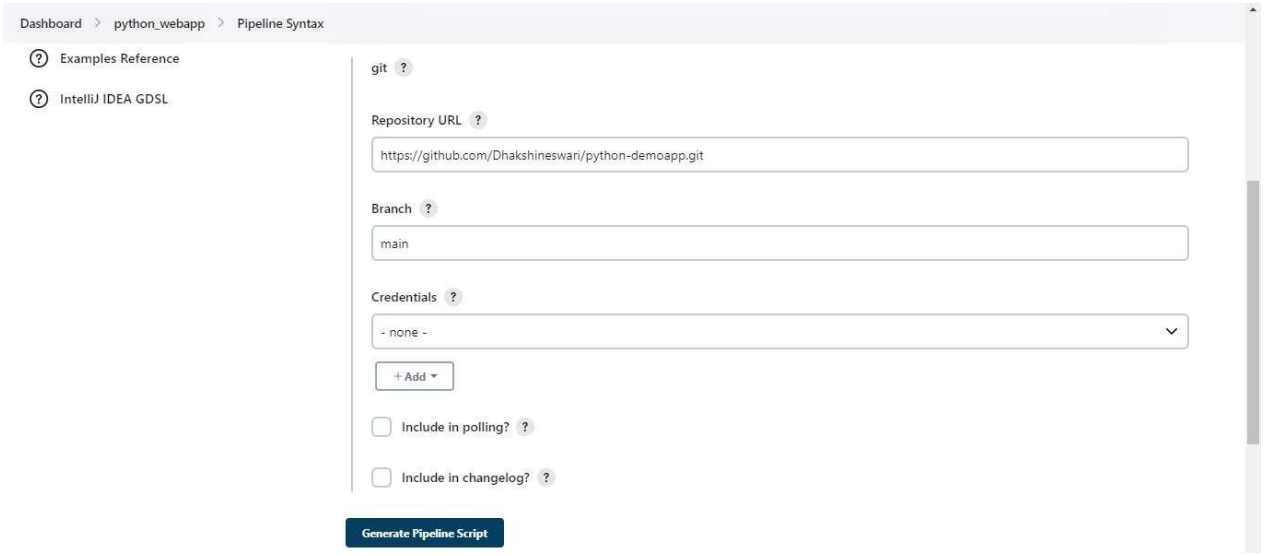


STEP 22: Click on pipeline syntax and Select the Snippet generator

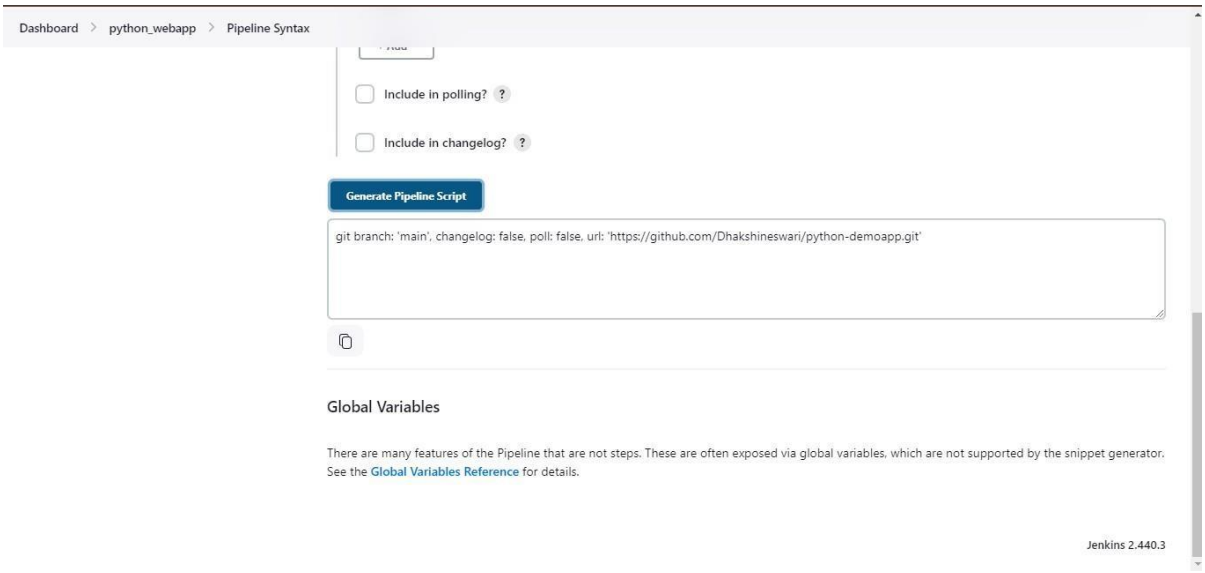
1. In the Sample step, choose the git: Git.



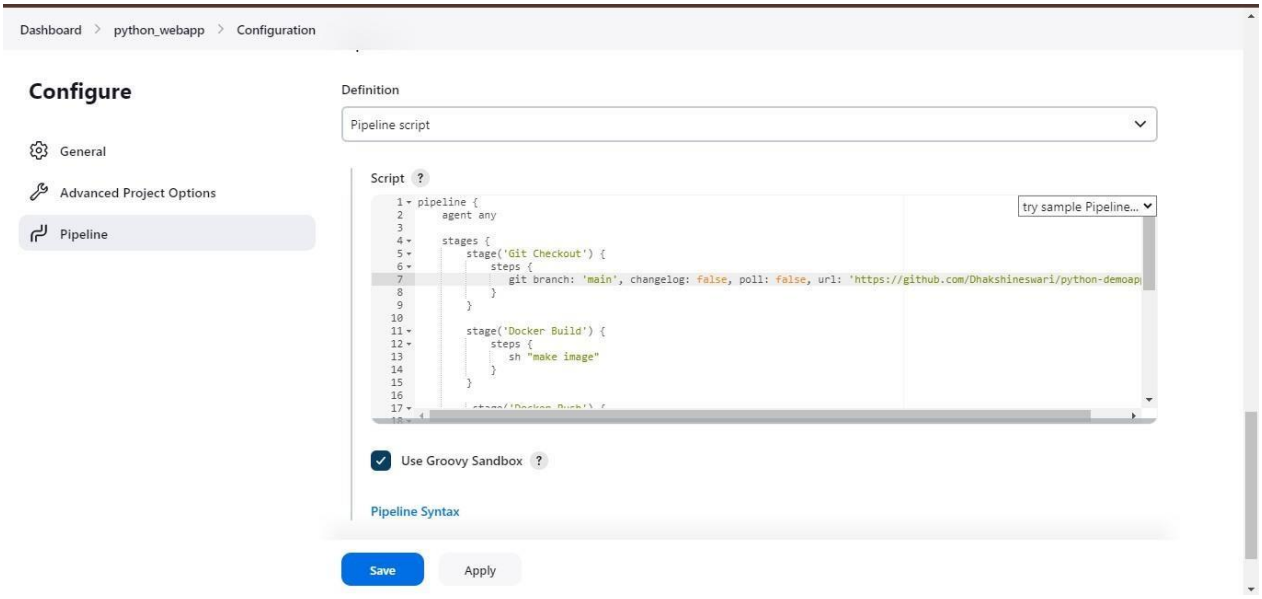
2. Paste your GitHub URL in the URL section.
3. Type as ‘main’ in Branch and deselect the options.



4. Click on Generate pipeline script.

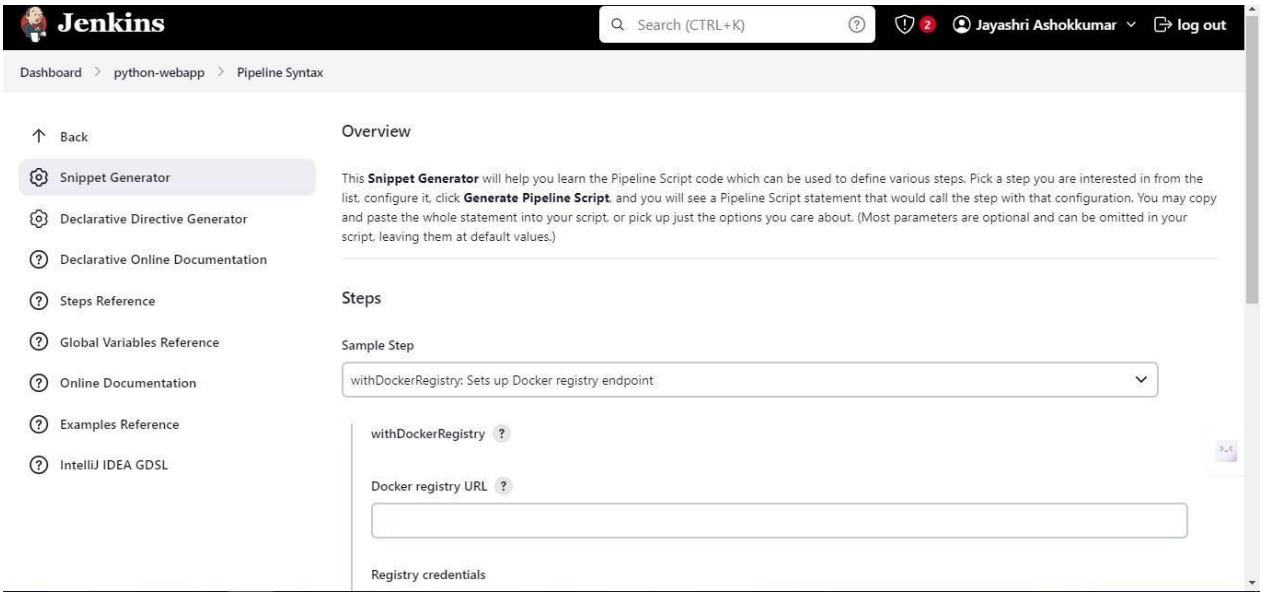


STEP 23: In Step 21, paste the above generator pipeline script in the URL portion of the code.

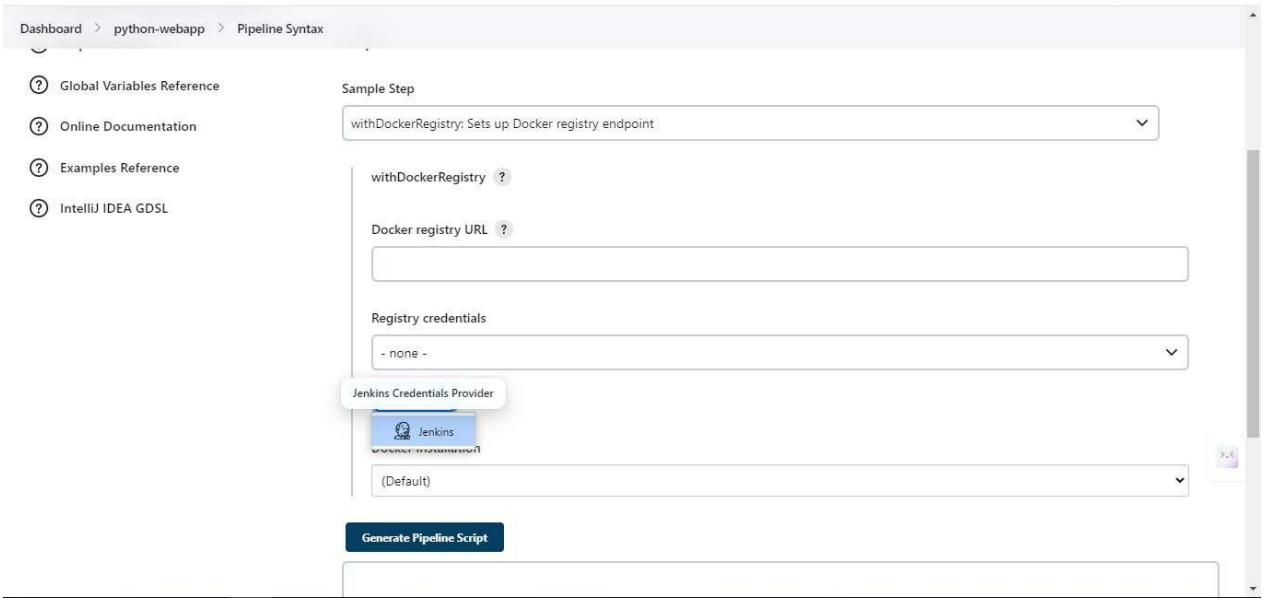


STEP 24: Again Click on pipeline and select the Snippet generator.

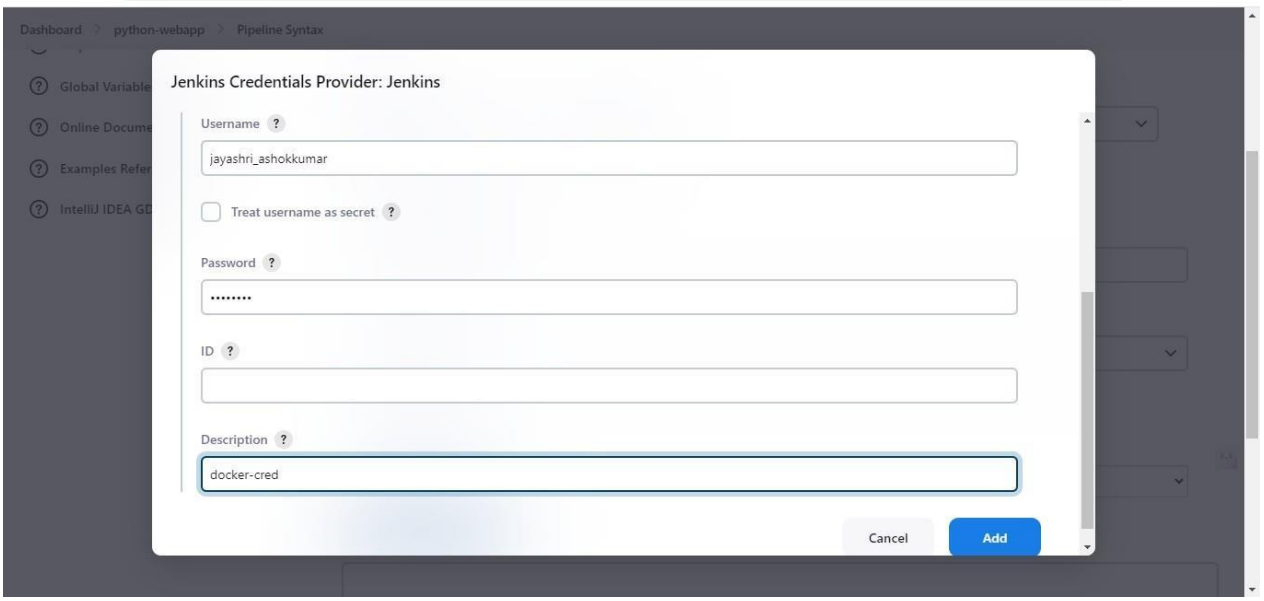
1. In the Sample step, choose the below option.



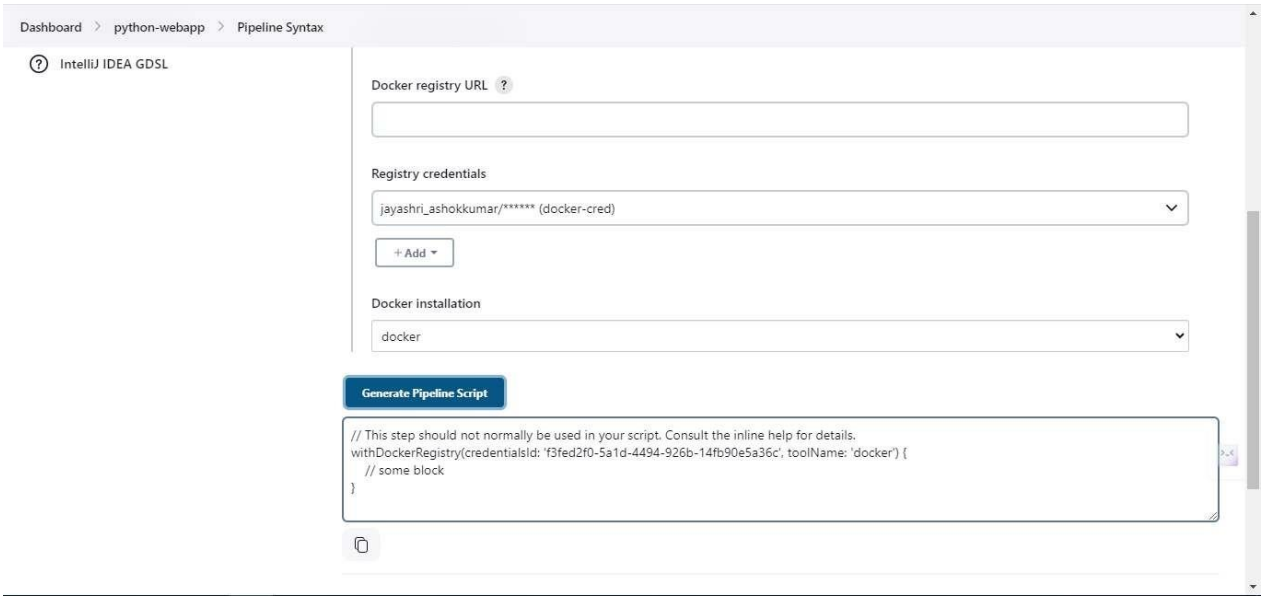
2. Make the Registry Credentials null and click Jenkins.



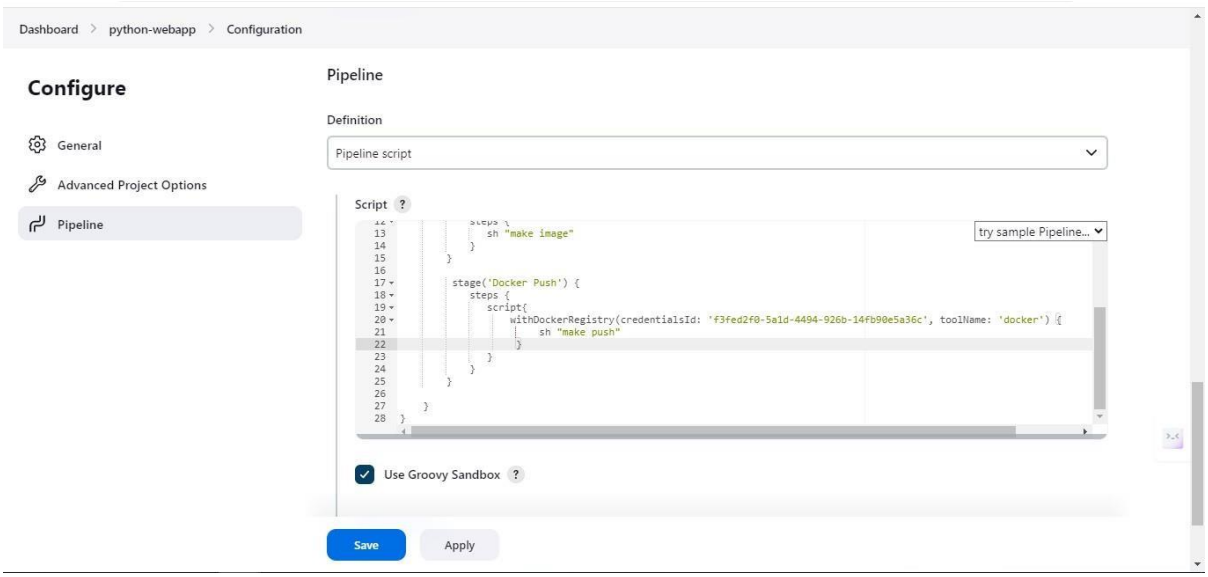
3. Now, provide the username and password (GitHub) and click save.



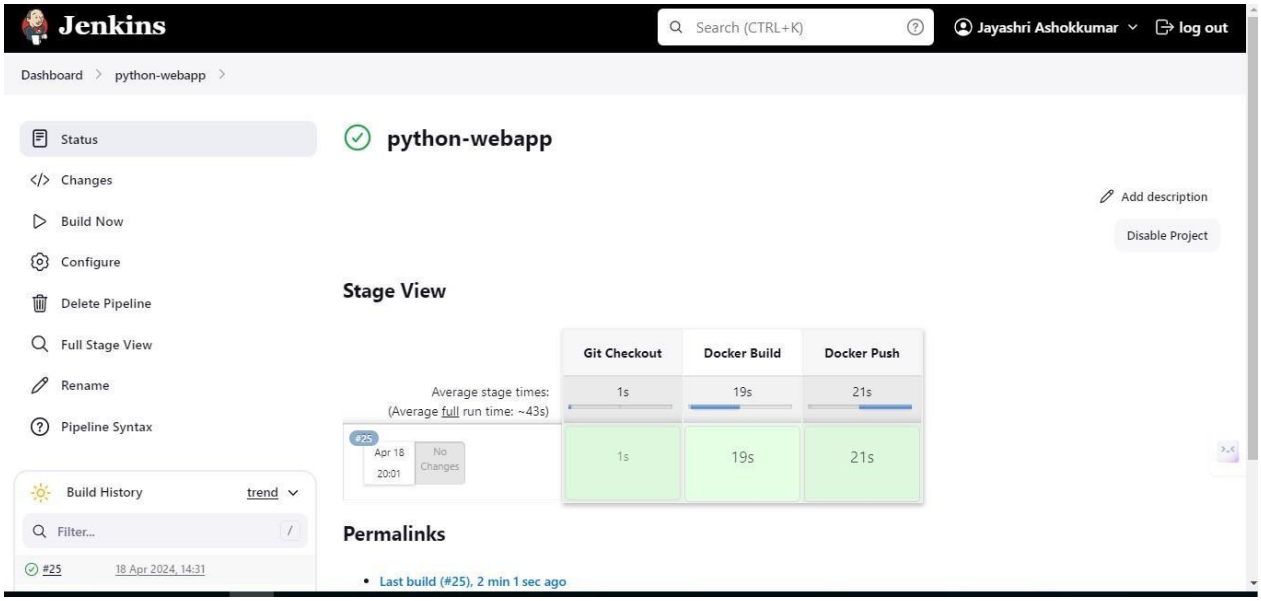
4. The Credentials will have been added in the Registry Credentials section select the docker below, then click on Generate pipeline script.



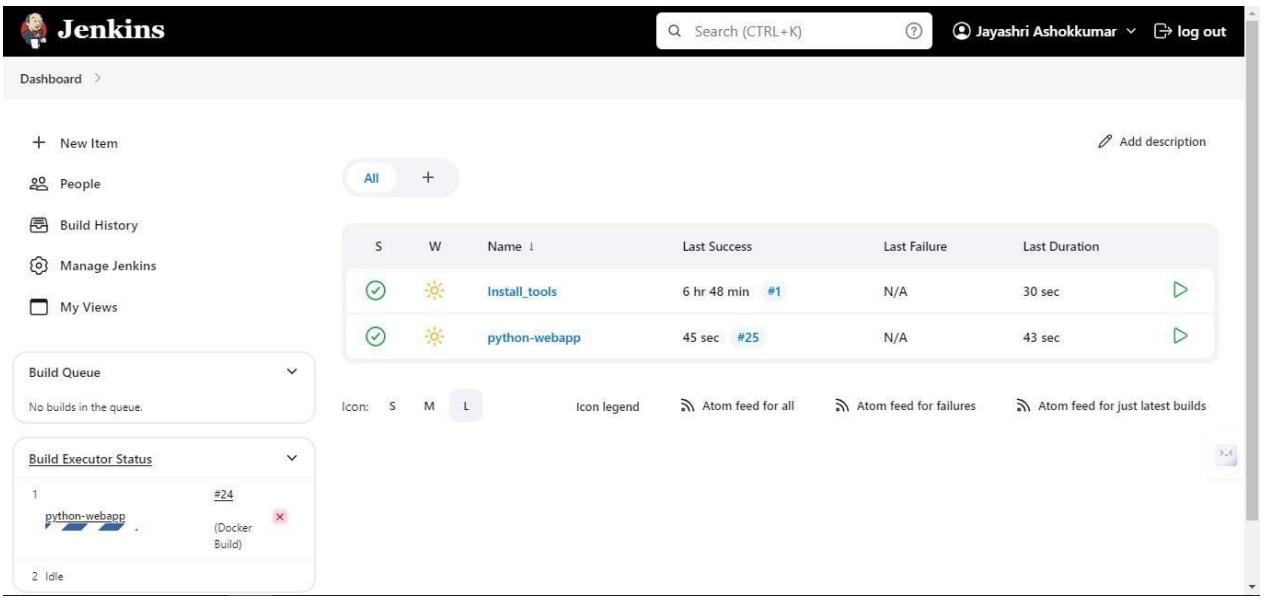
STEP 25: In Step 21, paste the above generator pipeline script in the Credentials ID and tool name portion of the code.



STEP 26: Click > dashboard > python-webapp > build now.



STEP 27: Now CI Pipeline has been built Successfully.



RESULT:

Thus, the CI Pipeline has been successfully created.

Ex. No: 5 CREATE A CD PIPELINE IN JENKINS AND DEPLOY IN CLOUD

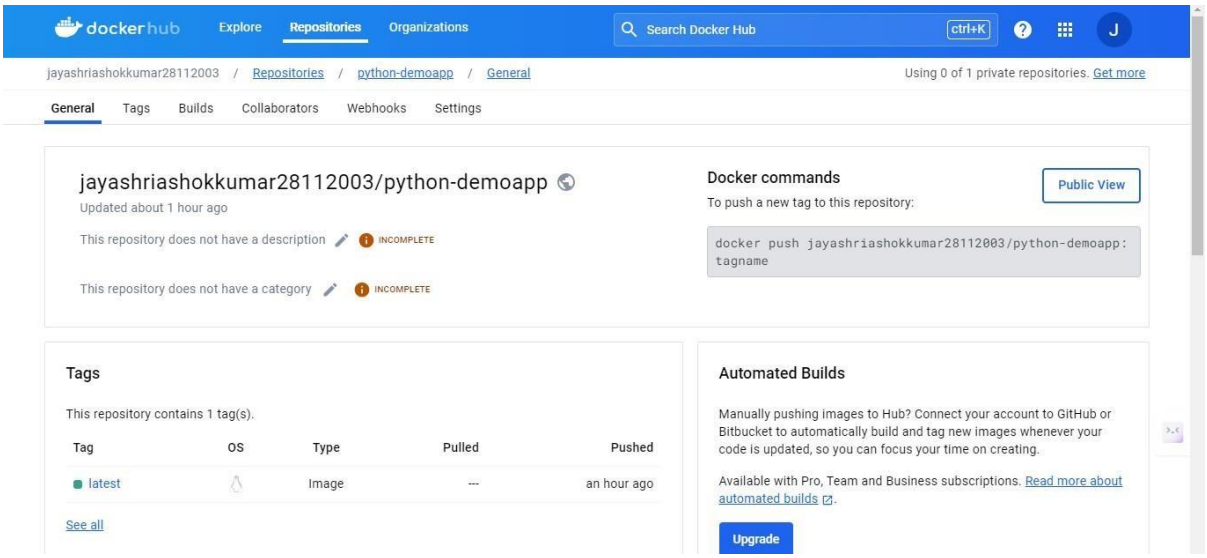
Date:

AIM:

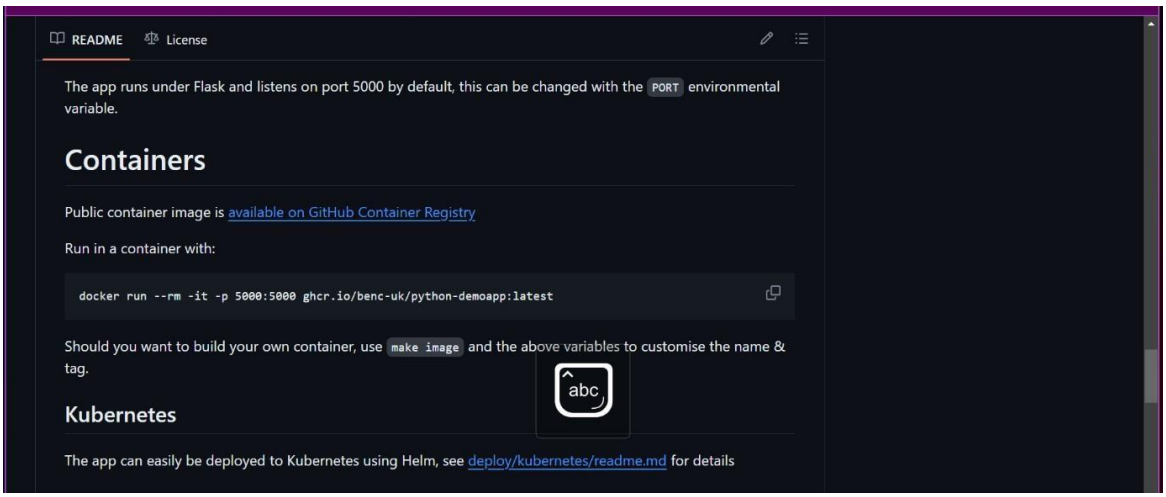
To create a CD pipeline in Jenkins and deploy it in the Cloud.

PROCEDURE:

STEP 1: Click Repositories > pythondemoapp > General. There, you will be able to see a command under Docker commands. Copy that command.

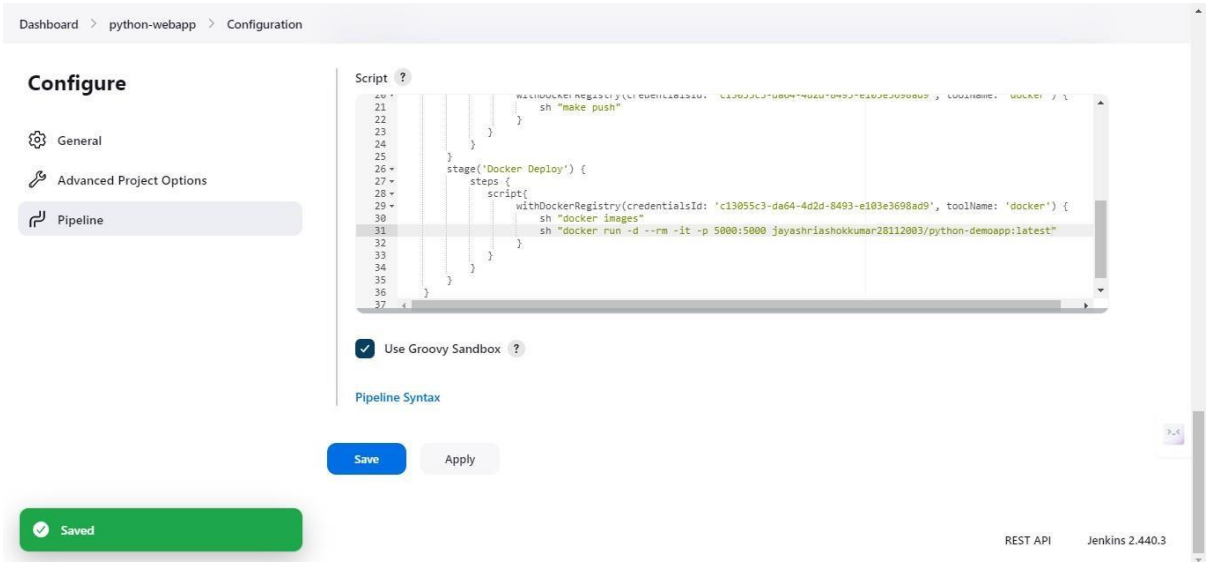


STEP 2: Now, go to your GitHub account. Scroll down to Containers and copy the command again for future use.

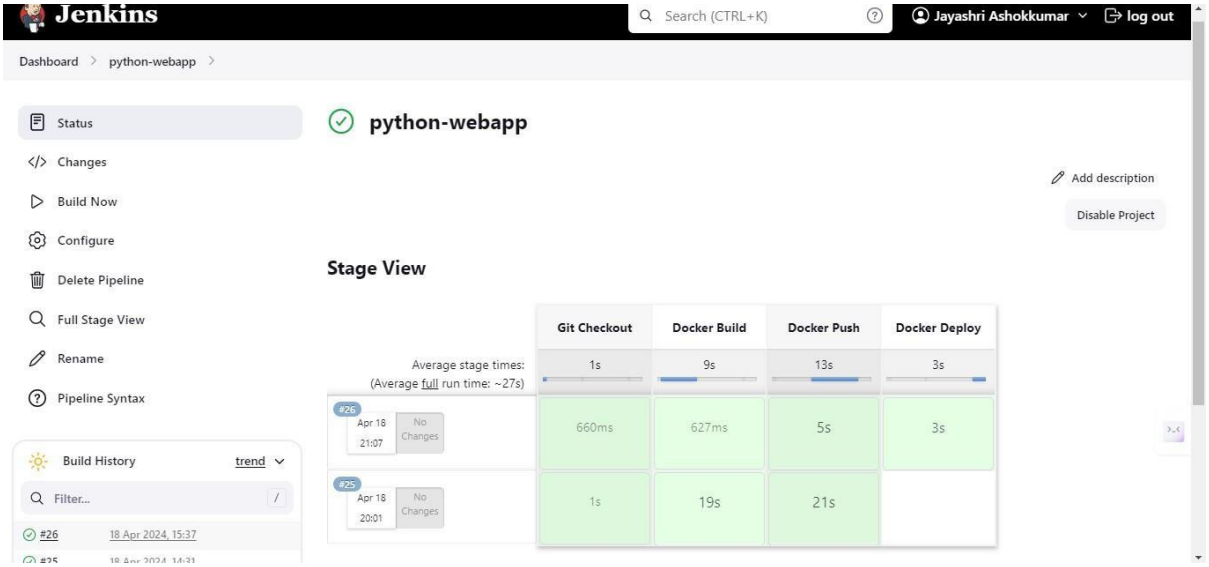


STEP 3: Now add the code (combination of the above commands) below to the code through which the CI Pipeline was built.

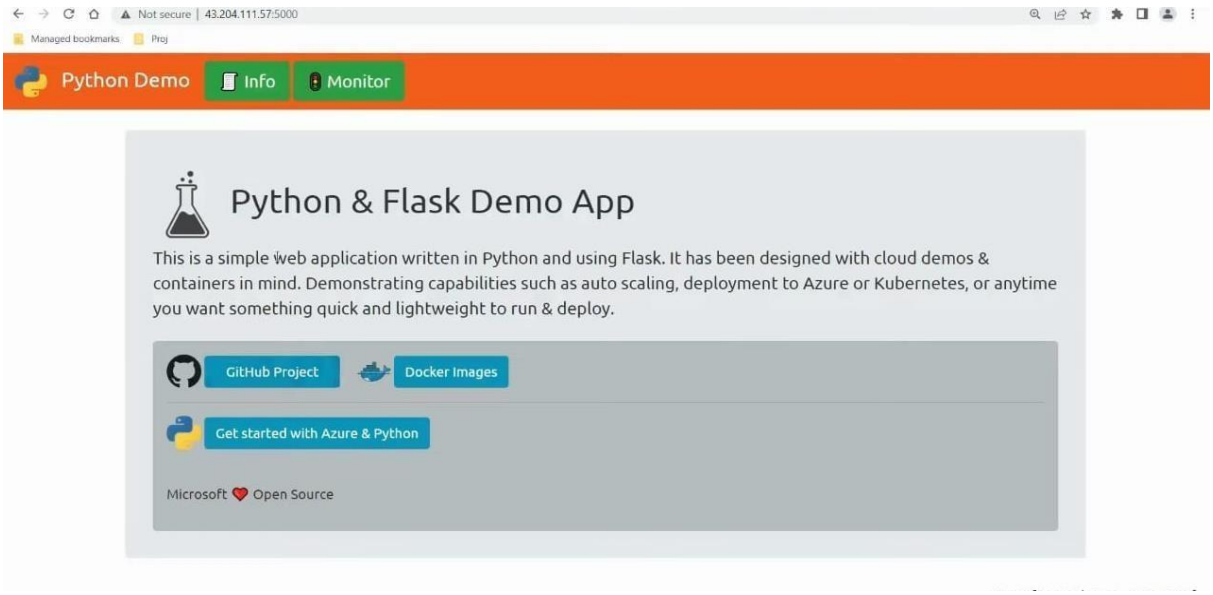
```
stage('Docker Deploy') {
        steps {
          script{
            withDockerRegistry(credentialsId: 'dbc54abf-de2b-42b8-a0d2-f5539c9b8997',
toolName: 'docker') {
                sh "docker images"
                sh "docker run -d --rm -it -p 80:5000 jayashriashokkumar28112003/python-
webapp:latest"
              }
          }
        }
      }
```



STEP 4: Click on Dashboard > python-webapp > Build Now



STEP 5: To access the Python web app copy and paste the public IP of the AWS instance and include port 5000 at the end of it. (For example: 41.204.111.57:5000)



RESULT:

Thus, the CD pipeline has been successfully created in Jenkins and deployed in the Cloud.

Ex. No: 6 **CREATE AN ANSIBLE PLAYBOOK FOR A SIMPLE WEB**

Date: **APPLICATION INFRASTRUCTURE**

AIM

To Create an Ansible Playbook for a simple web application infrastructure.

PROCEDURE

STEP 1: Create a directory named **ansible-practice1** and then navigate into that directory with the cd command.

The following commands are used to create a directory,
$ cd ~
$ mkdir ansible-practice1
$ cd ansible-practice1

To add content to playbook use the following commands,
$ nano playbook01.yml

```
ubuntu@ip-172-31-43-52:~/ansible-practice$ cd
ubuntu@ip-172-31-43-52:~$ mkdir ansible-practice1
ubuntu@ip-172-31-43-52:~$ cd ansible-practice1
ubuntu@ip-172-31-43-52:~/ansible-practice1$ nano playbook01.yml
ubuntu@ip-172-31-43-52:~/ansible-practice1$
```

STEP 2: Now create a playbook that contains the following content to be executed. The content to be added in the playbook **(playbook01.yml),**

```
---
- hosts: localhost
  tasks:
   - name: Print message
     debug:
     msg: Hello Ansible World
```

NOTE: The host name in the content must be changed to localhost.

```
 GNU nano 6.2                                playbook01.yml
---
- hosts: localhost
  tasks:
    - name: Print message
      debug:
        msg: Hello Ansible World




                                [ Wrote 6 lines ]
^G Help          ^O Write Out    ^W Where Is     ^K Cut          ^T Execute      ^C Location     M-U Undo
^X Exit          ^R Read File    ^\ Replace      ^U Paste        ^J Justify      ^/ Go To Line   M-E Redo
```

STEP 3: Run **ansible-playbook** with the same connection arguments, use an inventory file named **inventory** and the **sammy** user to connect to the remote server. The command to run the ansible playbook,

$ ansible-playbook -i inventory playbook-01.yml -u sammy

```
ubuntu@ip-172-31-43-52:~/ansible-practice1$ ansible-playbook -i inventory playbook01.yml -u sammy
[WARNING]: Unable to parse /home/ubuntu/ansible-practice1/inventory as an inventory source
[WARNING]: No inventory was parsed, only implicit localhost is available
[WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit localhost does not match
'all'

PLAY [localhost] ***************************************************************************************************

TASK [Gathering Facts] *********************************************************************************************
ok: [localhost]

TASK [Print message] ***********************************************************************************************
ok: [localhost] => {
    "msg": "Hello Ansible World"
}

PLAY RECAP *********************************************************************************************************
localhost                  : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

ubuntu@ip-172-31-43-52:~/ansible-practice1$
```

RESULT:

Thus, creating and executing an ansible playbook for a simple web application infrastructure has been executed successfully.

Ex. No: 7 **BUILD A SIMPLE APPLICATION USING GRADLE**

Date:

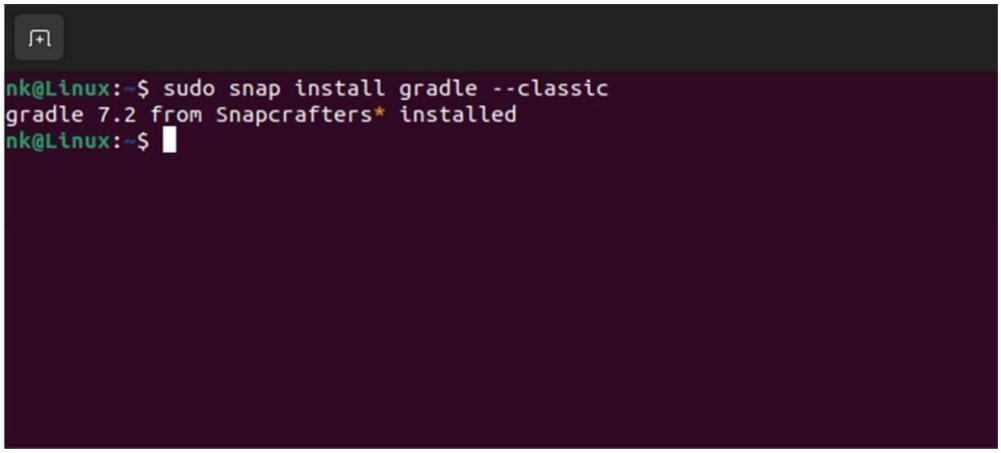
AIM

To build a simple application using Gradle.

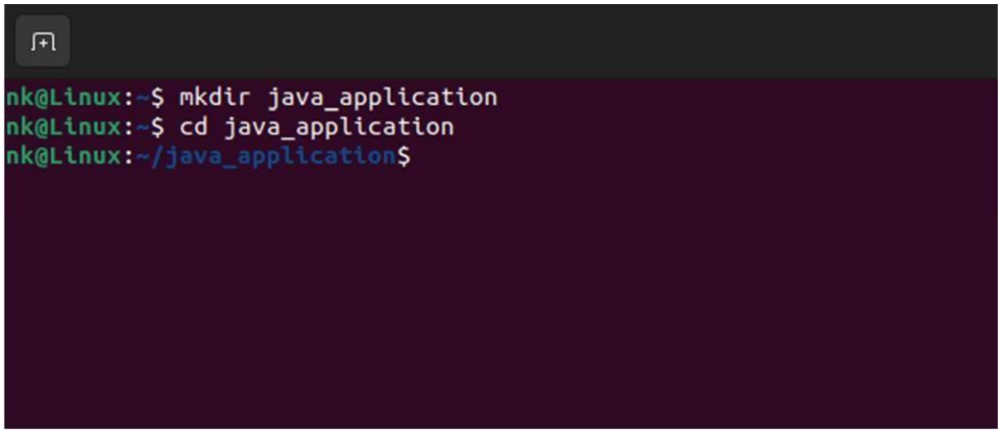
PROCEDURE

Step 1: Create directory: `java_application`.
Step 2: Change to directory: `java_application`.
Step 3: Run `gradle init` command.
Step 4: Select project type (application).
Step 5: Choose implementation language (Java).
Step 6: Select build script language (Groovy).
Step 7: Choose testing framework (JUnit 4).
Step 8: Enter project name.
Step 9: Enter source package.

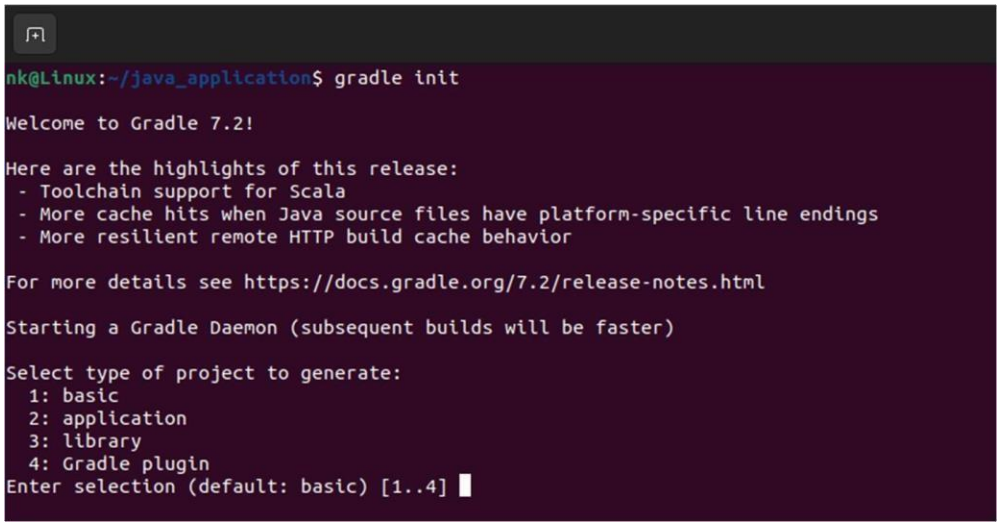
1. Install gradle using the following command



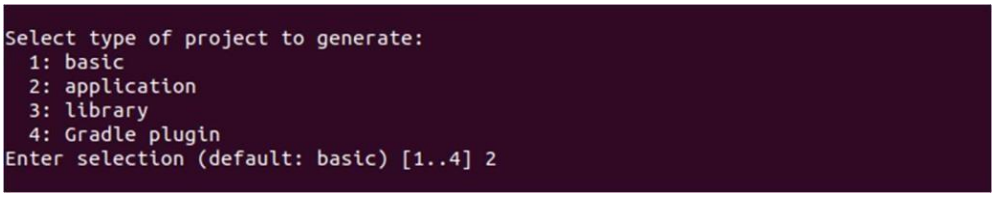
2. Create a directory named **java_application** and change directory:



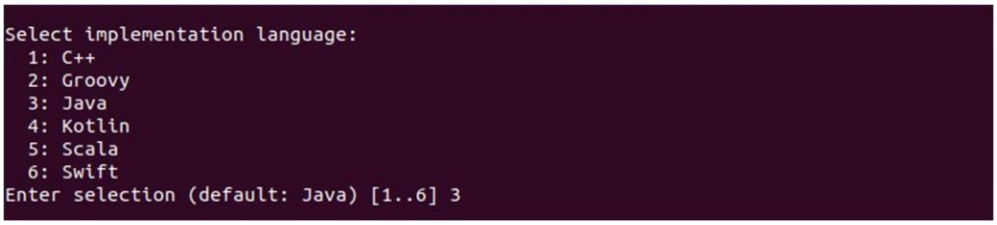
3. Run the **gradle init** command to create a new Gradle project



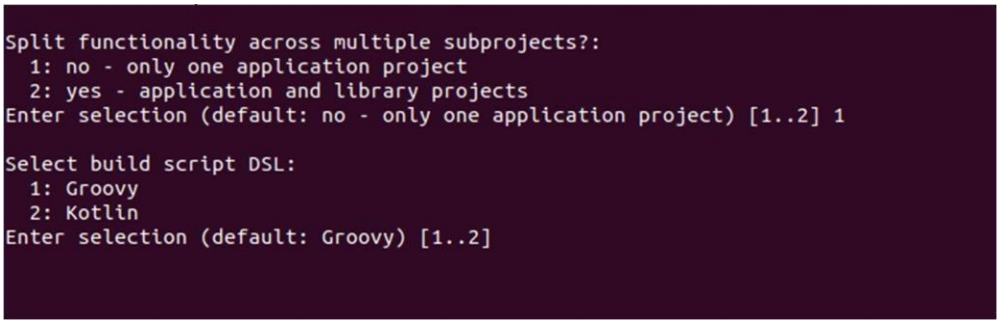
4. Select the project type (application, library, etc.) by typing `2` and pressing Enter for an application.



5. Choose the implementation language (Java, Kotlin, etc.) by typing `3` and pressing Enter for Java.

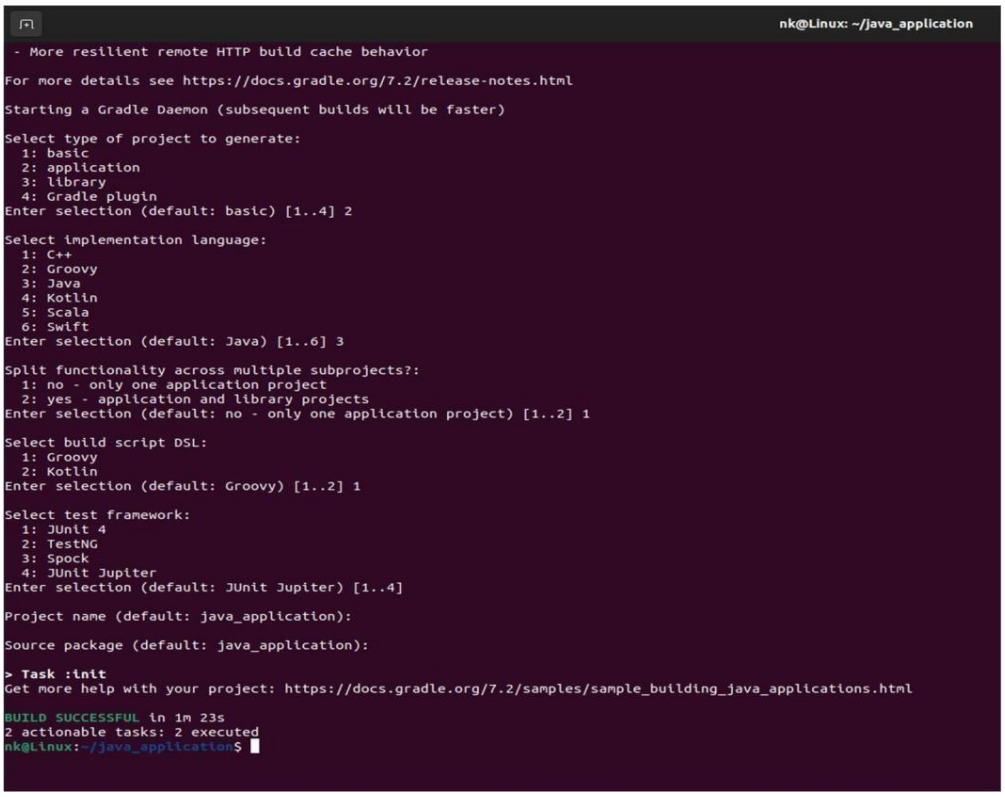


6. Select the default build script language (Groovy or Kotlin) by typing `1` and pressing Enter for Groovy.



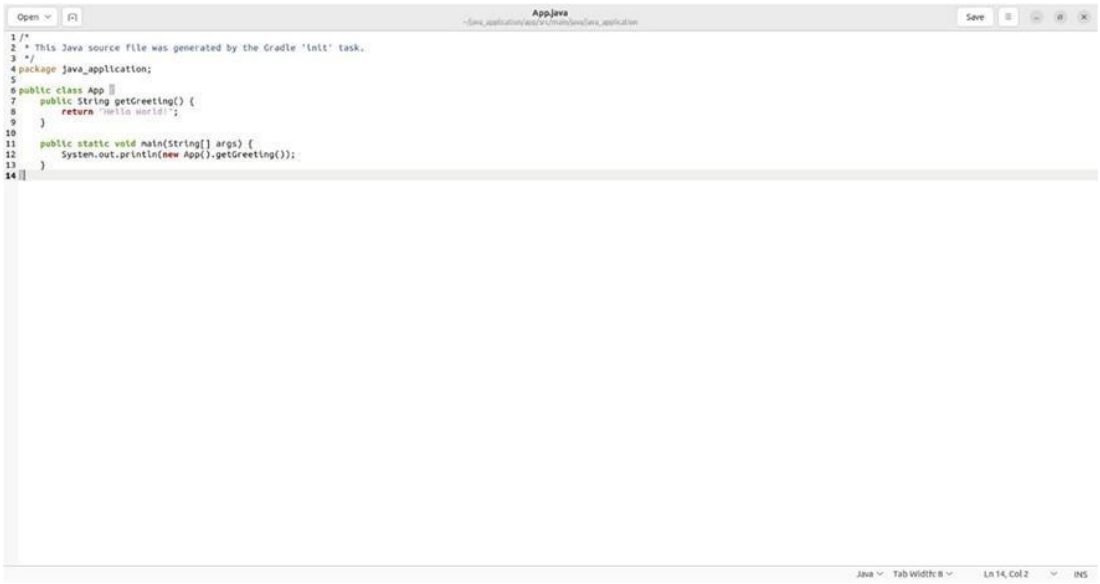
7. Choose the testing framework (JUnit 4, TestNG, etc.) if prompted. The default is Junit4

8. Enter the project name when prompted (default is the directory name) & Enter the source package when prompted (default is the directory name).



OUTPUT


Home / java_application / app / src / main / java / java_application
Recent
Starred
Home
Documents
Downloads
Music
Pictures
Videos
Trash
Other Locations
App.java
"App.java" selected (297 bytes)


Open
App.java
Save
1 /*
2 * This Java source file was generated by the Gradle 'init' task.
3 */
4 package java_application;
5
6 public class App {
7 public String getGreeting() {
8 return "Hello World!";
9 }
10
11 public static void main(String[] args) {
12 System.out.println(new App().getGreeting());
13 }
14 }
Java
Tab Width: 8
Ln 14, Col 2
INS

RESULT

Thus, the Java application was successfully created using Gradle.

Ex. No: 8 **INSTALL ANSIBLE AND CONFIGURE ANSIBLE ROLES AND TO**

Date: **WRITE PLAYBOOKS**

AIM

To install Ansible, configure Ansible roles and write playbooks.

PROCEDURE

Step 1: Refresh your system's package index so that it is aware of the packages available. Run the following command,

$ sudo apt update

```
*** System restart required ***
Last login: Fri Apr 12 13:39:11 2024 from 13.48.4.203
ubuntu@ip-172-31-43-52:~$ ansible --version
Command 'ansible' not found, but can be installed with:
sudo apt install ansible       # version 2.10.7+merged+base+2.10.8+dfsg-1, or
sudo apt install ansible-core  # version 2.12.0-1ubuntu0.1
ubuntu@ip-172-31-43-52:~$ sudo apt update
Hit:1 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Hit:2 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:3 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu jammy-security InRelease
Ign:5 https://pkg.jenkins.io/debian-stable binary/ InRelease
Hit:6 https://pkg.jenkins.io/debian-stable binary/ Release
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
2 packages can be upgraded. Run 'apt list --upgradable' to see them.
```

Step 2: The Ansible package and its dependencies are available in the default package repositories to install the latest Ansible version, add its ppa repository.

$ sudo apt upgrade

$ sudo apt install -y software-properties-common

$ sudo add-apt-repository --yes --update ppa:ansible/ansible

```
ubuntu@ip-172-31-43-52:~$ sudo apt upgrade
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
The following packages have been kept back:
  python3-update-manager update-manager-core
0 upgraded, 0 newly installed, 0 to remove and 2 not upgraded.
ubuntu@ip-172-31-43-52:~$ sudo apt install software-properties-common
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
software-properties-common is already the newest version (0.99.22.9).
0 upgraded, 0 newly installed, 0 to remove and 2 not upgraded.
```

```
ubuntu@ip-172-31-43-52:~$ sudo apt-add-repository --yes --update ppa:ansible/ansible
Repository: 'deb https://ppa.launchpadcontent.net/ansible/ansible/ubuntu/ jammy main'
Description:
Ansible is a radically simple IT automation platform that makes your applications and systems easier to deploy. Avoid w
riting scripts or custom code to deploy and update your applications— automate in a language that approaches plain Engl
ish, using SSH, with no agents to install on remote systems.

http://ansible.com/

If you face any issues while installing Ansible PPA, file an issue here:
https://github.com/ansible-community/ppa/issues
More info: https://launchpad.net/~ansible/+archive/ubuntu/ansible
Adding repository.
Adding deb entry to /etc/apt/sources.list.d/ansible-ubuntu-ansible-jammy.list
Adding disabled deb-src entry to /etc/apt/sources.list.d/ansible-ubuntu-ansible-jammy.list
Adding key to /etc/apt/trusted.gpg.d/ansible-ubuntu-ansible.gpg with fingerprint 6125E2A8C77F2818FB7BD15B93C4A3FD7BB9C3
67
Hit:1 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Hit:2 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:3 http://eu-north-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease
Ign:4 https://pkg.jenkins.io/debian-stable binary/ InRelease
Hit:5 https://pkg.jenkins.io/debian-stable binary/ Release
Hit:6 http://security.ubuntu.com/ubuntu jammy-security InRelease
Get:7 https://ppa.launchpadcontent.net/ansible/ansible/ubuntu jammy InRelease [18.0 kB]
Get:9 https://ppa.launchpadcontent.net/ansible/ansible/ubuntu jammy/main amd64 Packages [1128 B]
Get:10 https://ppa.launchpadcontent.net/ansible/ansible/ubuntu jammy/main Translation-en [752 B]
Fetched 19.9 kB in 1s (19.8 kB/s)
Reading package lists... Done
```

Step 3: Now install the ansible latest version using the command,

$ sudo apt install ansible

```
ubuntu@ip-172-31-43-52:~$ sudo apt install ansible
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
ansible is already the newest version (9.4.0-1ppa~jammy).
0 upgraded, 0 newly installed, 0 to remove and 2 not upgraded.
ubuntu@ip-172-31-43-52:~$
```

Now we are going to navigate the directory path to,

$ cd /etc/ansible/

Then, we will check the latest version of the ansible by,

$ ansible –version

```
ubuntu@ip-172-31-43-52:~$ cd /etc/ansible/
ubuntu@ip-172-31-43-52:/etc/ansible$ ansible --version
ansible [core 2.16.5]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/ubuntu/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /home/ubuntu/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0] (/usr/bin/python3)
  jinja version = 3.0.3
  libyaml = True
ubuntu@ip-172-31-43-52:/etc/ansible$
```

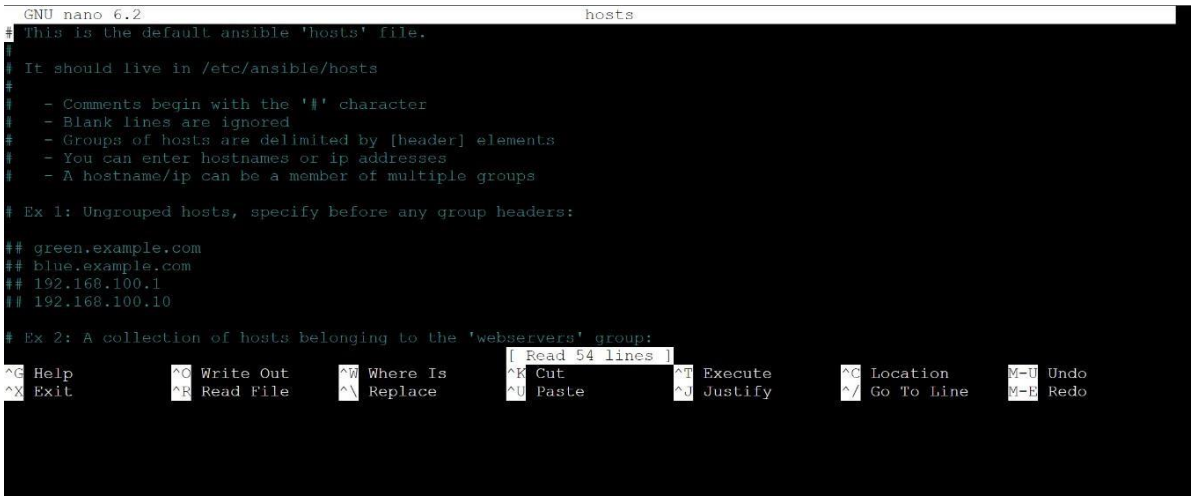
Step 4: To check the list of directories and files present in the ansible, run the command as,

$ ls –la

```
ubuntu@ip-172-31-43-52:/etc/ansible$ ls -la
total 20
drwxr-xr-x   3 root root 4096 Apr 12 13:51 .
drwxr-xr-x 103 root root 4096 Apr 12 13:51 ..
-rw-r--r--   1 root root  614 Mar 26 17:11 ansible.cfg
-rw-r--r--   1 root root 1175 Mar 26 17:11 hosts
drwxr-xr-x   2 root root 4096 Mar 26 17:11 roles
ubuntu@ip-172-31-43-52:/etc/ansible$
```

Then check the host file by,

$ sudo nano hosts



Step 5: Create an ansible role from scratch and run ansible-galaxy command.

$ ansible-galaxy init my-role

$ ls

```
ubuntu@ip-172-31-43-52:/etc/ansible$ sudo ansible-galaxy init role-name
- Role role-name was created successfully
ubuntu@ip-172-31-43-52:/etc/ansible$ ls
ansible.cfg  hosts  playbooks  role-name  roles
ubuntu@ip-172-31-43-52:/etc/ansible$
```

To view the role directory structure, run the tree command followed by the role name.

```
ubuntu@ip-172-31-43-52:/etc/ansible$ tree role-name/
role-name/
├── README.md
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml

8 directories, 8 files
ubuntu@ip-172-31-43-52:/etc/ansible$
```

Step 6: Create three ansible roles as follows:

The prerequisites role – Installs git.

The postgresql role – Installs postgresql service.

The apache role – Installs the Apache webserver.

Use the following commands to create the above,

$ sudo ansible-galaxy init prerequisites

$ sudo ansible-galaxy init PostgreSQL

$ sudo ansible-galaxy init apache

$ ls

```
ubuntu@ip-172-31-43-52:/etc/ansible$ sudo ansible-galaxy init prerequisites
- Role prerequisites was created successfully
ubuntu@ip-172-31-43-52:/etc/ansible$ sudo ansible-galaxy init PostgreSQL
- Role PostgreSQL was created successfully
ubuntu@ip-172-31-43-52:/etc/ansible$ sudo ansible-galaxy init apache
- Role apache was created successfully
ubuntu@ip-172-31-43-52:/etc/ansible$ ls
PostgreSQL  ansible.cfg  apache  hosts  playbooks  prerequisites  role-name  roles
ubuntu@ip-172-31-43-52:/etc/ansible$
```

Step 7: Define each role that you have created. To achieve this, you need to edit the main.yml file located in the 'tasks' folder for each role.

First for prerequisites,

$ cd prerequisites/tasks/

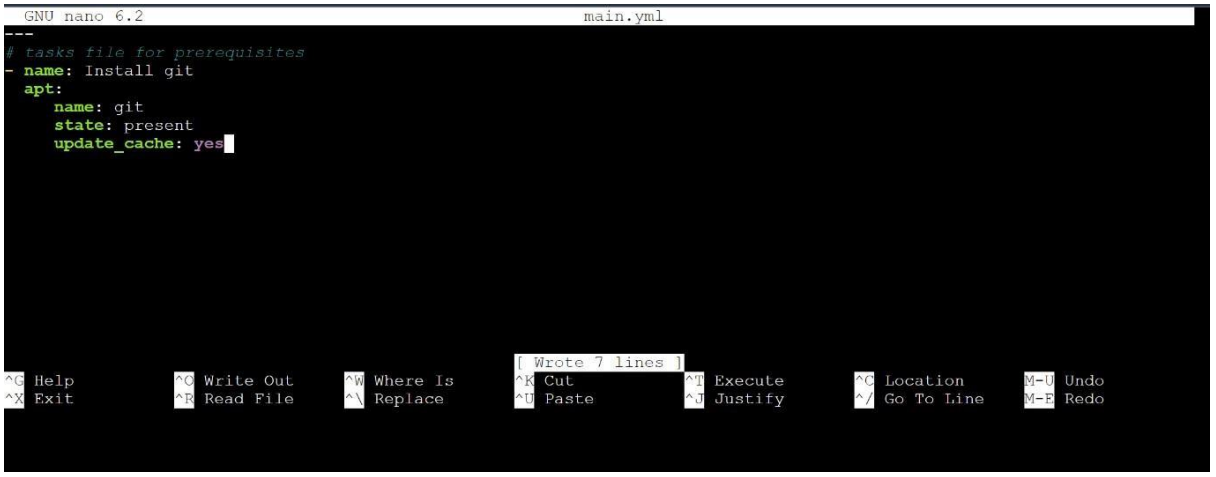
Then edit the main.yml file by the command,

$ sudo nano main.yml

```
ubuntu@ip-172-31-43-52:/etc/ansible$ cd prerequisites/tasks/
ubuntu@ip-172-31-43-52:/etc/ansible/prerequisites/tasks$ sudo nano main.yml
ubuntu@ip-172-31-43-52:/etc/ansible/prerequisites/tasks$
```

Then add the content to this main.yml file as

```
# tasks file for prerequisites
- name: Install git
  apt:
    name: git
    state: present
    update_cache: yes
```



Next for PostgreSQL,

$ cd PostgreSQL/tasks/

$ sudo nano main.yml

Then add the content to this main.yml file as,

tasks file for PostgreSQL

- name: Install PostgreSQL

apt:

name: postgresql

state: present

update_cache: yes

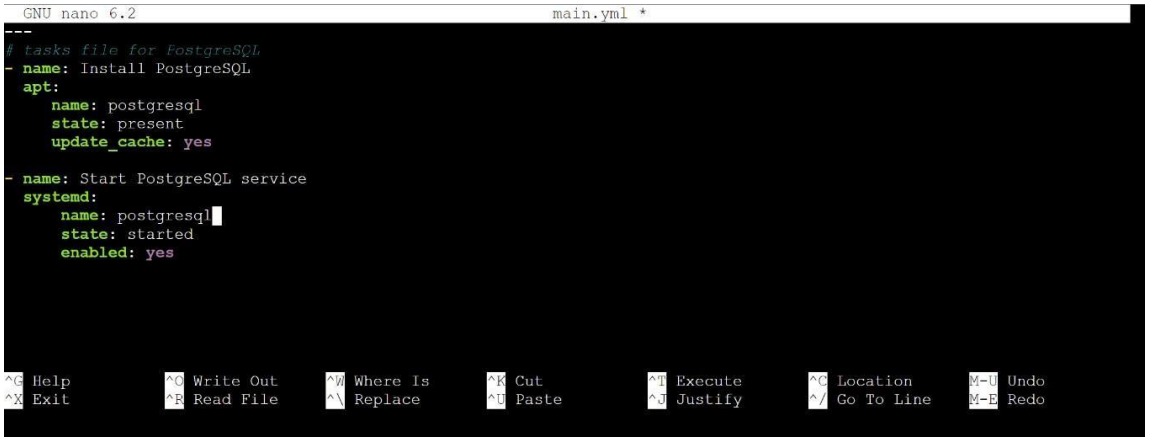
- name: Start PostgreSQL service

systemd:

name: postgresql

state: started

enabled: yes



Finally for the apache web server,

$ cd apache/tasks/

$ sudo nano main.yml

Then add the content to this main.yml file as,

tasks file for apache

- name: install Apache web server

  apt:

    name: apache2

    state: present

    update_cache: yes

Lastly, we are going to create a playbook file called stack.yml and call the roles by the command,

$ sudo nano stack.yml

Then add the content to this main.yml file as

- hosts: localhost

become: true

roles:

- prerequisites
- PostgreSQL
- apache

```
  GNU nano 6.2                                  stack.yml
---
- hosts: localhost
  become: true
  roles:
        -  prerequisites
        -  PostgreSQL
        -  apache

                                [ Wrote 7 lines ]
^G Help          ^O Write Out     ^W Where Is      ^K Cut           ^T Execute       ^C Location      M-U Undo
^X Exit          ^R Read File     ^\ Replace       ^U Paste         ^J Justify       ^/ Go To Line    M-E Redo
```

Step 8: Ensure that our roles are working as expected, and run the ansible playbook.

Run the following command,

$ sudo ansible-playbook stack.yml

```
ubuntu@ip-172-31-43-52:/etc/ansible$ sudo ansible-playbook stack.yml
[WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit localhost does not match
'all'

PLAY [localhost] ***************************************************************************************************

TASK [Gathering Facts] *********************************************************************************************
ok: [localhost]

TASK [prerequisites : Install git] *********************************************************************************
ok: [localhost]

TASK [PostgreSQL : Install PostgreSQL] *****************************************************************************
changed: [localhost]

TASK [PostgreSQL : Start PostgreSQL service] ***********************************************************************
ok: [localhost]

TASK [apache : install Apache web server] **************************************************************************
changed: [localhost]

PLAY RECAP *********************************************************************************************************
localhost                  : ok=5    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

ubuntu@ip-172-31-43-52:/etc/ansible$
```

Then check the versions to ensure that the packages were installed successfully,

$ git --version

$ apachectl –v

```
ubuntu@ip-172-31-43-52:/etc/ansible$ apachectl -v
Server version: Apache/2.4.52 (Ubuntu)
Server built:   2024-04-10T17:45:18
ubuntu@ip-172-31-43-52:/etc/ansible$ git --version
git version 2.34.1
ubuntu@ip-172-31-43-52:/etc/ansible$
```

RESULT

Thus, installing ansible, configuring ansible roles and writing playbooks have been executed successfully.